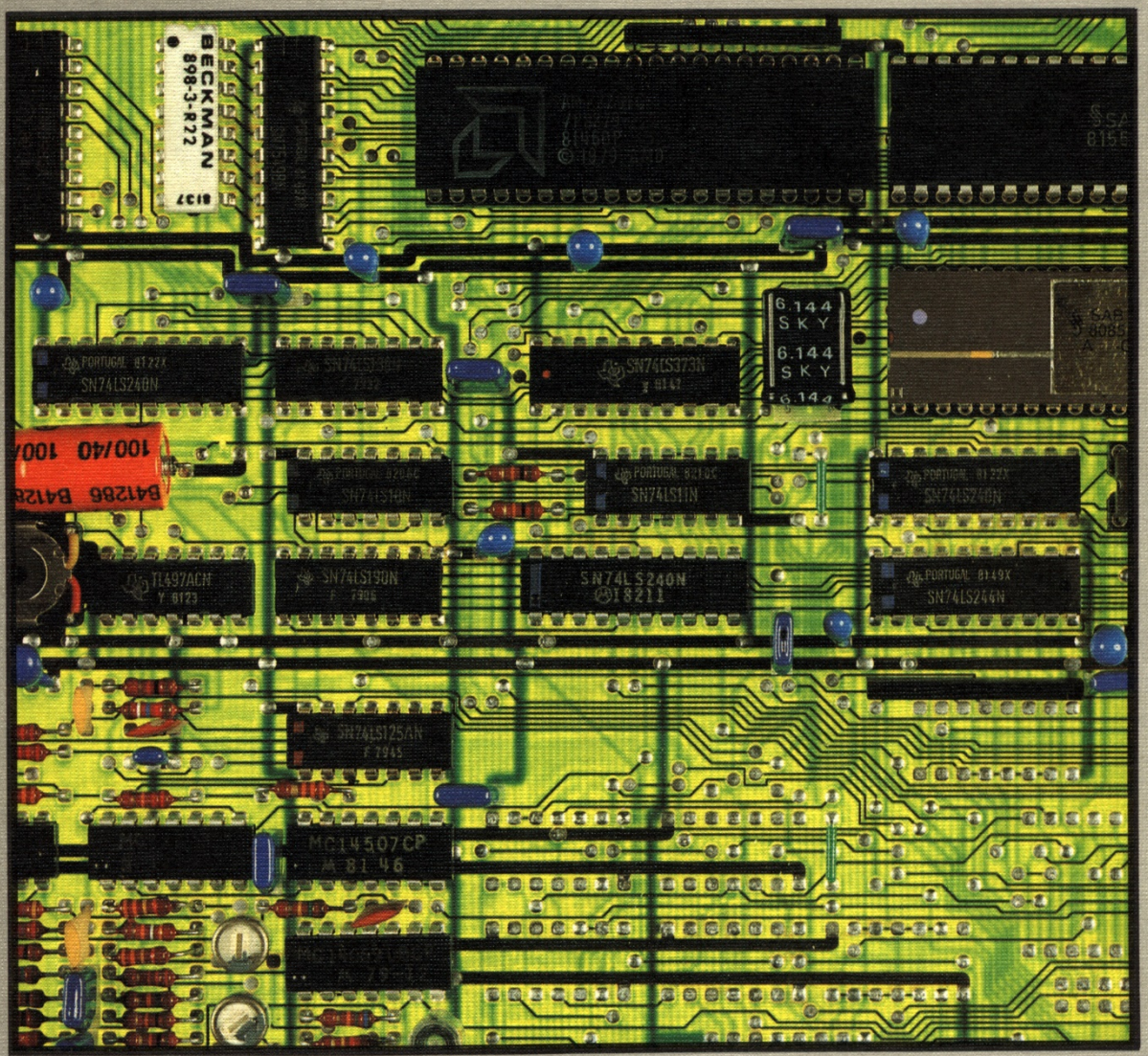
EXPERIMENTE

Ingrid Beck

Steuerungen mit dem Mikrocomputer



vgs

Ingrid Beck

Steuerungen mit dem Mikrocomputer

vgs Verlagsgesellschaft Schulfernsehen, Köln

CIP-Kurztitelaufnahme der Deutschen Bibliothek
Beck, Ingrid:
Steuerungen mit dem Mikrocomputer / Ingrid Beck.
1. Aufl. – Köln : Verlagsgesellschaft Schulfernsehen, 1985.
(Experimente)
ISBN 3-8025-1229-4

1. Auflage 1985
© Verlagsgesellschaft Schulfernsehen, Köln 1985
Zeichnungen: Peter Kohlöffel, Nonnenhorn
Satz: Fotosatz Froitzheim KG, Bonn
Druck und Bindung: Druckerei Laub GmbH & Co, Elztal-Dallau
Printed in Germany
ISBN 3-8025-1229-4

Inhalt

Teil 1: Allgemeines

1. Der prinzipielle Aufbau eines Mikrocomputers 7

1.1 Prozessor 7
1.2 Speicher 7
1.3 Bus-Struktur 8

2. Allgemeines zur Programmierung 9

3. Die wichtigsten Funktionseinheiten des ECB 85 12

3.1 Der Mikroprozessor 8085 A 18
3.1.1 Arithmetik-Logik-Einheit 18
3.1.2 Befehlsregister und Steuerlogik – Der Ablauf von Befehlen 18
3.1.3 Registerblock und Adressenpuffer 21
3.2 Der Speicher im ECB 85 22
3.3 Die Ein-/Ausgabe-Bausteine 8155 23
3.4 Die Anwender-Anschlüsse (Schnittstelle) des ECB 85 31

4. Zur Programmierung des 8085 A 32

4.1 Die Adressierungsarten 32
4.2 Das Prozessor-Statuswort PSW 33

5. Die Befehlsworte für den Mikroprozessor 8085 A 34

5.1 Transferbefehle für 8-Bit-Daten 34
5.2 Transferbefehle für 16-Bit-Daten 35
5.3 Arithmetische Operationen für 8-Bit-Daten 36
5.4 Arithmetische Operationen für 16-Bit-Daten 37
5.5 Logische Operationen: UND, ODER, EXOR, Negation 38
5.6 Vergleichs-Operationen 39
5.7 Rotations-Befehle 39
5.8 Sprungbefehle, Unterprogramme und Unterbrechungen 40
5.8.1 Sprungbefehle 40
5.8.2 Aufruf von Unterprogrammen (UP) 41
5.8.3 Programm-Unterbrechung 42

5.9 Die Befehle RIM und SIM 44
5.10 Der Befehl DAA 46
5.11 Befehle, die nur das Carry-Bit bearbeiten 46
5.12 Die Steuerbefehle HLT und NOP 46

6. Die Handhabung des Experimentier-Computers ECB 85 47

6.1 Der Aufbau des ECB 85 47
6.1.1 Das Anzeigefeld 47
6.1.2 Die Tastatur 50
6.2 Die Monitor-Kommandos 51

Teil 2: Übungen mit dem ECB 85

Vorbemerkung 62
Übung 1: Speicher-Kommandos 63
Übung 2: Ein erstes Programm 64
Übung 3: Zwei kleine Arithmetik-Programme 65
Übung 4: Der Ablauf arithmetischer Befehle mit Flagbyte-Auswertung 67
Übung 5: Ansteuerung von drei Leuchtdioden und Auswertung von drei Schalter-Signalen 71
Übung 6: Steuerung einer Verkehrsampel 74
Übung 7: Ansteuerung einer 7-Segment-Anzeige 79
Übung 8: Erzeugen von Tönen 83
Übung 9: Geschwindigkeits-Messung 89
Übung 10: Reaktionszeit-Messung 100
Übung 11: Der ECB 85 als Uhr 114
Übung 12: Ein Würfelprogramm 119
Übung 13: Der ECB 85 als Baustein-Prüfgerät 123

Teil 3: Anhang

A.1 Aufbau eines kleinen Mikrocomputersystems 126
A.2 Listing der in den Übungen benutzten Monitor-Unterprogramme des ECB 85 137
A.3 Quellenhinweise 139
A.4 Befehlsliste des 8085 A 140

Teil 1: Allgemeines

1. Der prinzipielle Aufbau eines Mikrocomputers

Die Bestandteile herkömmlicher Computer sind neben der Zentraleinheit (mit Steuerwerk, Rechenwerk, Arbeitsspeicher und Ein-/Ausgabe-Steuerung) vor allem Speichergeräte, Eingabe- und Ausgabegeräte sowie Dialoggeräte. Mikrocomputer werden in der Regel für eine spezielle Aufgabe eingesetzt, sehr häufig in Prozeß-Steuerungen und -Überwachungen. Dadurch sind einige Besonderheiten bedingt.

1.1 Prozessor

Die Zentraleinheit des Mikrocomputers heißt CPU (Central Processing Unit, zentrale Prozessor-Einheit). Sie enthält im Prinzip nur Steuer- und Rechenwerk, beinhaltet also gegenüber der Zentraleinheit großer Computer insbesondere nicht mehr den Arbeitsspeicher. In *Kapitel 3* wird die CPU des Experimentiercomputers ECB 85, d. h. der Prozessorbaustein SAB 8085 A, eingehend beschrieben.

1.2 Speicher

Eine der Möglichkeiten, die bei Computern verwendeten Speicher zu charakterisieren, ist die nach der Art des Zugriffs:
Bei *seriellem* Zugriff kann die gespeicherte Information nur in einer bestimmten Reihenfolge eingeschrieben oder abgerufen („ausgelesen") werden. Hierzu zählen vor allem die Magnetbandspeicher, bei Mikrocomputern meist mit Hilfe von Kompakt-Kassetten realisiert. Die zum Auffinden einer bestimmten Information benötigte Zeitspanne hängt hier sehr stark davon ab, an welcher Stelle des Bandes sich die Information befindet. Es muß – z. B. vom Bandanfang an – jede davorliegende Information sozusagen erst „übersprungen" bzw. darauf geprüft werden, ob sie schon die gewünschte ist.
Bei *wahlfreiem* Zugriff („random access", wörtl.: beliebiger Zugang) ist jeder Speicherplatz (s. u.) in gleicher Zeit erreichbar – unabhängig davon, auf welchem Platz vorher gelesen oder geschrieben wurde. Hier kann also eine bestimmte Zugriffszeit definiert werden; dies ist die Zeitspanne, die zum Aufsuchen und Lesen einer abgespeicherten Information und zum Übertragen ins Steuerwerk benötigt wird, wenn die Adresse (s. u.) bekannt ist.
In Mikrocomputer-Systemen werden hauptsächlich *Halbleiterspeicher* verwendet, die sich durch geringen Platzbedarf und kurze Arbeitszeiten auszeichnen. Man unterscheidet zwischen Schreib-/Lese-Speichern und Festwert-Speichern.
Die Schreib-/Lese-Speicher werden kurz mit RAM bezeichnet (Random Access Memory = Speicher mit wahlfreiem Zugriff, vgl. den vorigen Abschnitt). Beim Abschalten der Versorgungsspannung geht die gespeicherte Information meistens verloren; diese RAMs gehören zu den sogen. flüchtigen Speichern. Es gibt im wesentlichen zwei Arten von RAMs: die dynamischen und die statischen. Bei einem dynamischen RAM wird jedes Bit (siehe *Kapitel 2*) in Form einer Kondensator-Ladung gespeichert, die zyklisch immer wieder aufgefrischt werden muß. Die statischen RAMs bestehen im Prinzip aus Flip-Flops (auch bistabile Kippglieder genannt), die gesetzt bzw. nicht gesetzt sind.
Die Festwert- bzw. reinen Lesespeicher (ROM = Read Only Memory = Nur-Lese-Speicher) gehören zu den nichtflüchtigen Speichern; denn sie behalten auch ohne angelegte Versorgungsspannung die gespeicherte Information, die nicht geändert werden

soll; daher: read only. Genau genommen sind dies, wie die RAMs, auch Speicher mit wahlfreiem Zugriff; es ist jedoch üblich, lediglich die Eigenschaft des Nur-Lesens in der Bezeichnung ROM auszudrücken. In ein ROM werden bei der Herstellung einmal die Informationen eingeschrieben; man sagt auch, es wird „programmiert". Es gibt auch ROMs, die vom Anwender selbst zu programmieren sind, nämlich die PROMs (Programmable ROMs), und schließlich noch die EPROMs (Erasable PROMs), die vom Benutzer gelöscht werden können (to erase = ausradieren, löschen). Dieses Löschen geschieht mit UV-Licht. Danach können mit einer speziellen Einrichtung, die sich z. B. auch auf dem ECB 85 befindet, neue Informationen eingeschrieben werden. Bei der Benutzung in einem Mikrocomputer-System verhält sich ein EPROM genau wie ein ROM: Es kann also nur gelesen werden. Der Vollständigkeit halber seien noch die EEPROMs erwähnt (Electrical Erasable PROMs), die auf elektrischem Wege gelöscht werden können.

In Mikrocomputer-Systemen dient das RAM zum Aufnehmen der Informationen, die sich während des Programmablaufs ändern. Dagegen befindet sich das abzuarbeitende Programm mit bestimmten festen Werten, die dazu benötigt werden, meist im ROM bzw. EPROM. Bei der Programm-Entwicklung bzw. der Testphase wird das Programm jedoch im RAM abgelegt, damit es noch geändert oder verbessert werden kann. Erst danach wird es in ein EPROM übernommen; auf diesem Wege kann man sich eine „Programmbibliothek" aufbauen: Je nach eingesetztem EPROM erfüllt der Mikrocomputer verschiedene Aufgaben.

Die Organisation von Speichern

Es wurden schon die Begriffe „Adresse" und „Speicherplatz" erwähnt. Man kann den Speicher mit einem Stadtteil vergleichen, wie *Bild 1.1* zeigt. Jedem Speicherplatz wird eine Adresse zugeordnet, unter der er (und nur er) zu erreichen ist. Man beachte dabei, daß die Numerierung der Speicher-Plätze (= Speicher-Zellen) bei Null beginnt!

In jedem Speicherplatz befindet sich eine 8-Bit-Information, d. h. ein Datenwort der Länge 1 Byte; siehe dazu *Kapitel 2*. Viele Mikroprozessoren, so auch der im ECB 85 befindliche 8085, verfügen über Adressen mit einer Länge von 16 Bit, können also Speicher mit 2^{16} = 65 536 Speicherplätzen ansprechen. Man sagt auch, der Speicher hat eine Kapazität von 64 K (= 64*1 024). Die Zahl 2^{10} = 1024 wird dabei mit K abgekürzt – nicht zu verwechseln mit k = Kilo-... (= 1 000).

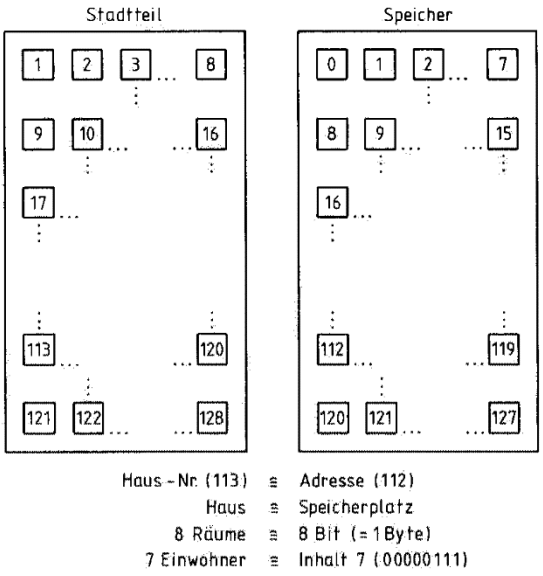


Bild 1.1: Zur Organisation von Speichern.

1.3. Bus-Struktur

Charakteristisch für den Aufbau der Mikrocomputer ist die Bus-Struktur für die Übertragung von Daten, Speicheradressen und Steuersignalen; sie ermöglicht es, die unterschiedlichsten, für die jeweilige Spezialaufgabe benötigten Peripherie-Bausteine an den Prozessor anzuschließen. Das Bus-System besteht aus drei Sammelschienen (engl. bus); sie verbinden die einzelnen Bausteine bzw. Funktionseinheiten eines Mikrocomputersystems miteinander.

Im Blockschaltbild eines Mikrocomputersystems ist die Bus-Struktur gut zu erkennen: *Bild 1.2*. Die in *Kapitel 3* zu besprechenden E/A-Bausteine synchronisieren die Arbeit des Mikroprozessors mit den peripheren Geräten.

Daten-Bus: Über ihn werden die Daten zwischen Mikroprozessor und Speicher bzw. E/A-Bausteinen über mehrere Leitungen bidirektional, d. h. wahlweise in eine der beiden Richtungen, übertragen. Im Falle des Mikroprozessors 8085, der 8 Bit parallel verarbeitet, wird folglich ein 8-Bit-Datenbus verwendet, d. h. der Datenbus hat 8 Leitungen.

Adreß-Bus: Er überträgt die Adresse des gerade anzusprechenden Speicherplatzes oder E/A-Kanals; vgl. *Kapitel 3*. Wenn der Prozessor, wie der 8085, einen 64-K-Speicher ansteuern kann, werden 16 Adreß-Leitungen benötigt; es liegt also ein 16-Bit-Adreßbus vor. Hier ist eine Besonderheit zu verzeichnen: Bei manchen Mikroprozessoren, so auch beim 8085, werden zum Einsparen von Anschlußpins die niederwertigen 8 Bit der Adresse und die Daten „gemultiplext", d. h. über dieselben 8 Leitungen des Adreß/Daten-Bus geführt. Der 8-Bit-Adreß/Daten-Bus überträgt also zu einem bestimmten Zeitpunkt das niederwertige Byte der Adresse und zu einem anderen Zeitpunkt ein Datenbyte. Das Signal ALE des Steuerbus (s. u.) zeigt dabei an, ob gerade ein Datenbyte oder das niederwertige Byte einer Adresse übertragen wird; durch das Multiplexen werden somit insgesamt 7 Anschlußpins bzw. Leitungen eingespart.

Steuer-Bus: Über seine Leitungen werden u. a. Signale übertragen, die den Speicher- oder E/A-Bausteinen mitteilen, ob in ihnen gerade gelesen oder geschrieben werden soll. Auch die oben erwähnte ALE-Leitung (Adressenspeicher-Freigabe) gehört zum Steuerbus.

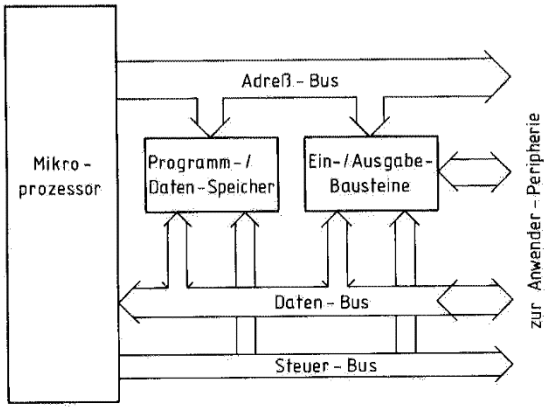


Bild 1.2: Das Blockschaltbild eines Mikrocomputersystems.

2. Allgemeines zur Programmierung

Mikrocomputer-Schaltungen funktionieren nach den Gesetzen der Digitaltechnik: Es kommen – einmal abgesehen vom (hochohmigen) Tristate – nur zwei Spannungspegel vor, die man mit H (für high) und L (für low) bezeichnet. Ebenso können diesen zwei möglichen Zuständen die Ziffern 1 und 0 zugeordnet werden. Diese 1/0- oder Ja/Nein-Entscheidung ist sozusagen die kleinste Einheit der Information; man bezeichnet sie als ein *Bit*. Eine Gesamtheit von 8 Bit nennt man ganz allgemein ein *Byte*. Diese Bezeichnungsweise ist praktisch, weil in den Mikrocomputern die Bits der vorkommenden Informationen nicht einzeln nacheinander verarbeitet werden, sondern parallel jeweils in Gruppen zu 8 oder 16 Bit (selten auch 4 Bit). Diese Anzahl heißt auch *Wortlänge*, d. h. ein Datenwort beispielsweise eines 8-Bit-Mikrocomputers hat 8 Bit, also gerade 1 Byte; dies ist beim Prozessor 8085 im ECB 85 der Fall.

Weil, wie oben gesagt, für die Darstellung der Informationen nur die Ziffern 0 und 1 benutzt werden, spricht man auch von einem binären System (binär = aus 2 Einheiten bestehend). Anders ausgedrückt: jedes Byte kann als 8stellige Dualzahl angesehen werden, z. B.:

00000011 (Zahlenwert drei)
11011001 (Zahlenwert zweihundertsiebzehn)
11111111 (Zahlenwert zweihundertfünfundfünfzig).

Aber in den Mikrocomputern werden nicht nur Daten (d. h. beispielsweise Zahlenwerte zu Beginn oder am Ende einer Berechnung, oder auch Signale, die an Peripherie-Geräte abgegeben werden, usw.), sondern es werden auch Speicherplatz-Adressen und Befehle (siehe *Kapitel 4*) in diesem binären System dargestellt bzw. verarbeitet. Man kann daher jeden Computer im Prinzip durch Eingabe der Befehle in binärer Form, d. h. in der sog. *Maschinensprache*, programmieren. Man spricht auch vom *Objektcode* oder vom *Maschinencode*.

In diesem Code besteht jeder Befehl für den Mikroprozessor 8085 aus 1 bis 3 Bytes, so daß ein ganz einfaches und kurzes Programmbeispiel, das weiter unten erläutert wird, so aussieht:

0011 1110 0001 1000
1000 0010
0100 1111
0111 0110

Das Eintippen einer solchen u. U. recht langen Folge von Nullen und Einsen ist jedoch sehr mühsam,

zeitraubend und fehleranfällig. Eine erste Vereinfachung für den Benutzer besteht darin, das Programm nicht im Dualcode, sondern im *Hexadezimal-Code* (abgekürzt: *Hex-Code*) einzugeben. Ohne weitere Erläuterung der zugrunde liegenden Gesetzmäßigkeiten sind zur Verdeutlichung des hexadezimalen Zahlensystems bzw. des Hexadezimal-Codes in *Bild 2.1* die ersten 16 Zahlen (von null bis fünfzehn) tabellarisch wiedergegeben.

Für die Darstellung von 1 Byte braucht man also zwei Hex-Zeichen; so entspricht z. B. der binären Darstellung 11011001 die Hex-Zahl D9. Man spricht auch von zwei *Halb-Bytes*, hier D und 9.

Das oben binär geschriebene Programm sieht im Hex-Code so aus:

```
3E  18
82
4F
76
```

Beim ECB 85 sind Befehle, Daten und Adressen in dieser Form anzugeben, denn die Umcodierung (hexadezimal zu binär) wird vom Betriebsprogramm des Computers übernommen. Die hexadezimale Schreibweise des Maschinencodes ist, wie man sieht, schon wesentlich kürzer und übersichtlicher als die binäre. Sie hat aber immer noch den großen Nachteil, daß man die Bedeutung der einzelnen Programmzeilen nicht ohne weiteres erkennt – es sei denn, man weiß die Codierung aller möglichen Befehle des Prozessors auswendig (beim 8085 sind das 246).

dezimal	dual	hexadezimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Bild 2.1: Die Zahlen null bis fünfzehn im Dezimal-, im Dual- und im Hexadezimal-Code.

Deshalb verfaßt der Benutzer sein Programm in aller Regel auf einem sogen. „höheren Niveau"; damit ist ein Code bzw. eine Programmsprache gemeint, die sich mehr oder weniger eng an Elemente der menschlichen, meist der englischen, Sprache anlehnt. Das verglichen mit dem Maschinencode nächsthöhere Niveau ist die sogen. *Assembler-Sprache*, oft einfach *Assembler* genannt. Häufig wird auch die Bezeichnung *mnemo-Code* benutzt. Das griechische Wort *Mneme* bedeutet Gedächtnis; damit soll angedeutet werden, daß man sich die Bedeutung der einzelnen Befehle in diesem Code leichter merken kann. Das kleine Beispiel-Programm sieht nun so aus:

```
MVI  A, 18H
ADD  D
MOV  C,A
HLT
```

Und tatsächlich kann man z. B. in MOV unschwer das englische Wort für bewegen (to move) erkennen; es werden also Daten transportiert. ADD bedeutet entsprechend addieren (to add), usw. Mit Hilfe dieser Programmsprache Assembler kann der Benutzer sein Programm (das sogen. „Quellprogramm", s. u.) wesentlich leichter erstellen als in hexadezimaler oder gar binärer Form. Er muß die Übersetzung in den Maschinencode, das *Assemblieren*, zuweilen selbst „von Hand" vornehmen. Dies nennt man „Hand-Assemblierung". Für das schon gezeigte kleine Programmbeispiel sieht die vollständige Tabelle mit dem fertigen Programm dann üblicherweise so aus wie in *Bild 2.2* dargestellt. Die Bedeutung der Spalten „Adresse" und „Label" wird später besprochen, und die beim 8085 möglichen Befehlsworte werden in *Kapitel 5* behandelt.

Bei den in diesem Buch beschriebenen Experimenten werden die Programme also im Mnemo-Code verfaßt („Quellprogramm", s. u.) und dann, wie in der Tabelle von *Bild 2.2* gezeigt, in den hexadezimalen Maschinencode übersetzt, der dann in den ECB 85 eingetippt wird.

Für sehr viele Mikrocomputer gibt es fertig erhältliche oder sogar schon im Gerät implementierte Programme, die die Übersetzung vom Mnemo-Code in den Maschinencode übernehmen. Diese Programme nennt man Assembler. Auch der Mnemo-Code selbst wird – wie schon gesagt – oft als Assembler bezeichnet. Man muß daher darauf achten, ob das Übersetzungsprogramm oder die Programmsprache gemeint ist.

Die Assemblersprache ist für das Programmieren

Adresse	Hex-Code	Label	Mnemo-Code	Kommentar
1800	3E 18	Start:	MVI A, 18H	Bringe 18H (= 24D) in den Akkumulator
1802	82		ADD D	Addiere zum Akku-Inhalt den Inhalt des Registers D und speichere das Ergebnis im Akku
1803	4F		MOV C,A	Bringe Akku-Inhalt in das Register C
1804	76		HLT	Halte an

Bild 2.2: Beispiel für eine Programmtabelle. Zur Bedeutung von Akkumulator und Registern siehe *Kapitel 3.1.*

Quellprogramm	Bearbeitung durch	Endprodukt
maschinennahe Sprachen:		
Maschinen-Code	—	Maschinen-Code
Mnemo-Code	Assembler (evtl. von Hand)	Maschinen-Code
höhere Progr.-Sprachen:		
z.B. BASIC	Interpreter (zeilenweise) oder Compiler	Maschinen-Code
z.B. ALGOL, FORTRAN	Compiler (ges. Programm auf einmal)	Maschinen-Code

Bild 2.3: Eine Übersicht über einige Programmier-Sprachen und ihre Umsetzung in den Maschinen-Code.

von Steuerungs- und Überwachungsroutinen sowie für die Erstellung von Betriebssystem-Programmen geeignet.
Eine bei Home- und Personal-Computern, die mit Mikroprozessoren arbeiten, sehr verbreitete höhere Programmiersprache ist das BASIC („Beginner's All Purpose Symbolic Instruction Code", etwa: „symbolischer Allzweck-Befehls-Code für Anfänger"). Hier besorgt in der Regel ein sogen. *Interpreter* (engl., Übersetzer) die Umwandlung in den Maschinencode und die Programmausführung. Er interpretiert das Programm Zeile für Zeile jedesmal direkt vor der Ausführung. Dadurch ist der Ablauf von BASIC-Programmen recht langsam; vor allem wird dies in Programmschleifen deutlich, die in Computerprogrammen häufig auftreten. Denn jede Anweisung wird jedesmal, wenn sie wieder auftritt, stets von neuem durch Unterprogrammsprünge ausgeführt.
Die an etwas größeren Computern verwendeten höheren Programmiersprachen wie z.B. ALGOL, FORTRAN, Cobol, etc. werden in einem sogen. *Compiler* (engl., Bearbeiter) ein einziges Mal vor dem ersten Start übersetzt; beim eigentlichen Programmablauf wird dann nur noch ein direkt im Maschinencode vorliegendes Programm abgearbeitet.
Ein vom Programmierer geschriebenes Programm bezeichnet man oft als *Quellprogramm* (auch Quellenprogramm oder Quellcode). Damit sind nun alle in diesem Zusammenhang wichtigen Begriffe bzgl. des Programmierens von Computern bzw. Mikrocomputern definiert. *Bild 2.3* verdeutlicht noch einmal die Zusammenhänge.
An dieser Stelle der bewußt sehr knapp gehaltenen allgemeinen Einführung soll noch auf zwei sehr wichtige Gesichtspunkte der Informations-Darstellung und -verarbeitung hingewiesen werden.
Erstens muß, falls Zweifel möglich sind, immer klar erkennbar sein, in welchem Code eine Information (bzw. in welchem Zahlensystem eine Zahl) angegeben ist. Der erste Befehl im oben geschilderten Programmbeispiel lautete im Mnemo-Code MVI A, 18H. Dabei besagt das „H", daß hier die Hexadezimal-Zahl 18 vorliegt (gesprochen: „eins-acht"). Die Basis des Hexadezimalsystems ist 16 (nicht 10 wie beim gewohnten Dezimalsystem), so daß 18H gleichbedeutend ist mit 24D (D für „dezimal"), denn $1*16+8=24$. Theoretisch könnte man schreiben: MVI A, 24D. Die Kennzeichnung B für binär (bzw.

dual) angegebene Zahlen wird selten nötig sein, da eine Ziffernfolge wie 00011100 praktisch immer als Dualzahl bzw. Information im Binärcode gedeutet wird.

Zweitens kann ein 8-Bit-Wort (1 Byte, meist hexadezimal in 2 Halb-Bytes geschrieben) verschiedenartige Bedeutungen haben, je nachdem, wo es auftritt. Auch darüber muß sich der Programmierer stets im klaren sein. Das erste Byte im kleinen Beispielprogramm lautete 3E (binär: 00111110); und weil es in diesem Falle das allererste Byte eines Programms ist, muß es ein sogen. *Befehlsbyte,* d. h. eine Anweisung an den Prozessor sein (denn sonst „würde sich ja nichts tun"). Man spricht auch von einem *Op-Code (Operations-Code),* der hier befiehlt: „Bringe das nachfolgend angegebene Byte in den Akkumulator". Natürlich könnte 3E auch einfach einen Zahlenwert bzw. ein *Datenbyte* repräsentieren, nämlich wenn es beispielsweise als zweites Byte im obigen Programm vorkäme. Andererseits könnte es auch (in einem anderen Programmzusammenhang) ein *Adreßbyte* bedeuten. Eine dritte Möglichkeit, die hexadezimal als 3E angegebene Information zu deuten, ist die als *ASCII-Zeichen* (ASCII = American Standard Code for Information Interchange). Dieser Code ist ein 7-Bit-Code; er wird in Mikrocomputern zur Darstellung von Buchstaben, Ziffern und Sonderzeichen benutzt. Jedes Zeichen entspricht dann einem Byte, wobei das höchstwertige Bit stets 0 ist. 3E bedeutet hier: > (d. h. „größer als"), stellt also ein sogen. Sonderzeichen dar.

Hier noch einmal übersichtlich zusammengefaßt die möglichen Bedeutungen des Datenbytes 00111110:

als Zahl:	dual 00111110
	dezimal 62
	hexadez. 3E
als Op-Code:	MVI A,
als ASCII-Zeichen:	> („größer als")

3. Die wichtigsten Funktionseinheiten des ECB 85

In Anlehnung an die Blockschaltbild-Darstellung eines Mikrocomputersystems *(Bild 1.2)* werden in diesem Kapitel folgende Teile des ECB 85 behandelt:

Prozessor	(Baustein SAB 8085 A)
Speicher	(EPROM: 1 oder 2 SAB 2716, RAM: 2 AM 9124 sowie der RAM-Teil des SAB 8155)
E/A-Bausteine	(1 oder 2 SAB 8155)

Die Buchstaben-Kombination SAB in der Typenbezeichnung der Bausteine ist herstellerbezogen (hier: Siemens) und gibt vor allem Auskunft über die Herstellungsart und den zulässigen Temperaturbereich. Der Übersichtlichkeit halber wird im folgenden nur die einfache Typennummer verwendet (8085 A, 8155 etc.).

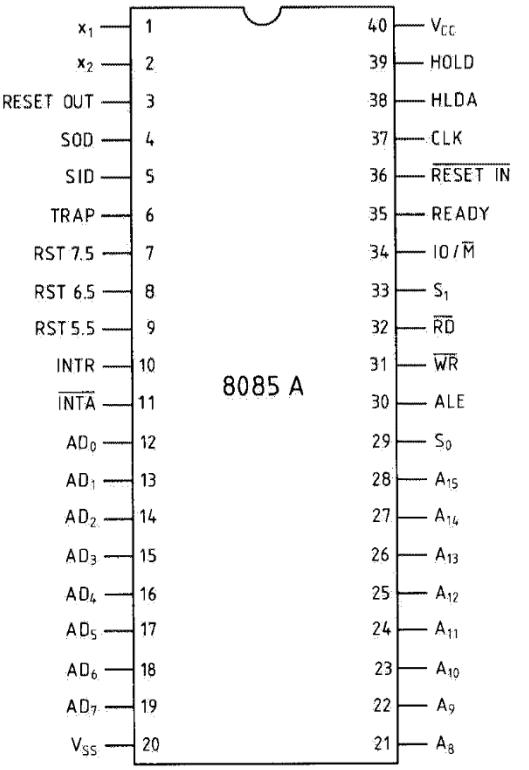


Bild 3.1.1: Der Anschlußplan des Mikroprozessor-Bausteins 8085 A.

<table>
<tr><th>Bezeichnung (Symbol)</th><th>Nummer</th><th>Eingang (E) Ausgang (A)</th><th>Funktion</th></tr>
<tr><td>A_8–A_{15}</td><td>21–28</td><td>A</td><td>Adressen-Bus: Die höherwertigen 8 Bits der Speicheradresse oder die 8 Bits der E/A-Adresse; die Ausgänge sind während der Betriebszustände »Hold«, »Halt« und »Reset« hochohmig.</td></tr>
<tr><td>AD_0–AD_7</td><td>12–19</td><td>E/A</td><td>Gemultiplexter Adressen-/Daten-Bus: Während der ersten Taktperiode (T-State) eines Maschinenzyklus stehen am Bus die niederwertigen 8 Bit der Speicheradresse (oder die E/A-Adresse) an. In der zweiten und dritten Taktperiode wird der Bus anschließend zum Daten-Bus.</td></tr>
<tr><td>ALE</td><td>30</td><td>A</td><td>Adressenspeicher-Freigabe: Das Signal tritt in der ersten Taktperiode eines Maschinenzyklus auf und ermöglicht, daß die Adresse in den integrierten Adressenspeichern von Peripherie-Bausteinen gespeichert wird. Die fallende Flanke des ALE-Signals ist so eingestellt, daß die Aufbau- und Haltezeiten der Adressen-information garantiert sind. Die fallende Flanke von ALE kann auch zum Takten der Status-Information verwendet werden. Der Ausgang ALE wird niemals hochohmig.</td></tr>
<tr><td>S_0, S_1 und IO/$\overline{M}$</td><td>29, 33 34</td><td>A</td><td>Maschinenstatus
<table><tr><th>IO/$\overline{M}$</th><th>S_1</th><th>S_0</th><th>Status</th></tr><tr><td>0</td><td>0</td><td>1</td><td>Speicher schreiben</td></tr><tr><td>0</td><td>1</td><td>0</td><td>Speicher lesen</td></tr><tr><td>1</td><td>0</td><td>1</td><td>E/A schreiben</td></tr><tr><td>1</td><td>1</td><td>0</td><td>E/A lesen</td></tr><tr><td>0</td><td>1</td><td>1</td><td>Operationscode-Abruf</td></tr><tr><td>1</td><td>1</td><td>1</td><td>Interrupt-Quittung</td></tr><tr><td>*</td><td>0</td><td>0</td><td>Halt</td></tr><tr><td>*</td><td>X</td><td>X</td><td>Hold</td></tr><tr><td>*</td><td>X</td><td>X</td><td>Reset</td></tr></table>* = hochohmig (Tri-state)
X = undefiniert
S_1 kann als voreilendes R/$\overline{W}$-Statussignal verwendet werden. IO/$\overline{M}$, S_0 und S_1 werden am Anfang eines Maschinenzyklus gültig und bleiben es über den gesamten Zyklus. Die abfallende Flanke des ALE-Signals kann zum Speichern der Zustände dieser Leitungen verwendet werden.</td></tr>
<tr><td>$\overline{RD}$</td><td>32</td><td>A</td><td>Lesen (Read): Ein L-Signal an $\overline{RD}$ gibt an, daß der ausgewählte Speicher- oder E/A-Baustein gelesen werden soll und daß der Daten-Bus für die Datenübertragung zur Verfügung steht. Der Ausgang ist in den Betriebsarten »Hold«, »Halt« und »Reset« hochohmig.</td></tr>
</table>

Bild 3.1.2: Die Anschlüsse des Prozessor-Bausteins 8085 A und ihre Funktion, *Teil 1.*

Bezeichnung (Symbol)	Nummer	Eingang (E) Ausgang (A)	Funktion
$\overline{WR}$	31	A	Schreiben (Write): Ein L-Signal an $\overline{WR}$ gibt an, daß die am Daten-Bus anstehenden Daten in den angewählten Speicherplatz oder E/A-Kanal geschrieben werden sollen. Die Daten werden mit der steigenden Flanke von $\overline{WR}$ übergeben. Während der Betriebsarten »Hold«, »Halt« und »Reset« ist der Ausgang hochohmig.
READY	35	E	Ein H-Signal an READY während eines Lese- oder Schreibzyklus zeigt an, daß der Speicher oder der Peripherie-Baustein zum Ausgeben oder Aufnehmen von Daten bereit ist. Wenn der READY-Eingang L-Signal führt, wartet die Zentraleinheit über eine ganze Zahl von Taktperioden, bis READY auf H-Pegel übergeht, worauf der Lese- oder Schreibzyklus beendet wird.
HOLD	39	E	HOLD zeigt an, daß eine andere Einheit den Adressen- und Daten-Bus anfordert. Nach dem Eintreffen der HOLD-Anforderung gibt die Zentraleinheit den Bus frei, sobald die augenblicklich ablaufende Bus-Operation beendet ist. Interne Operationen können weiterhin ablaufen. Der Mikroprozessor hat erst dann wieder Zugang zum Bus, wenn das Signal HOLD zurückgenommen wird. Sobald das HOLD-Signal quittiert ist, werden die Adressen- und Daten-Leitungen sowie $\overline{RD}$, $\overline{WR}$ und IO/$\overline{M}$ hochohmig.
HLDA	38	A	Quittung von HOLD: Das Signal zeigt an, daß die Zentraleinheit eine HOLD-Anforderung empfangen hat und den Bus in der nächsten Taktperiode freigegeben wird. HLDA geht auf L-Signal über, sobald die HOLD-Anforderung zurückgenommen ist. Die Zentraleinheit übernimmt den Bus eine halbe Taktperiode nach dem Übergang von HLDA auf L-Pegel.
INTR	10	E	Unterbrechungs-Anforderung: Der Eingang wird als allgemeiner Interrupteingang verwendet. Er wird nur während der vorletzten Taktperiode eines Befehls abgefragt sowie während der Zustände »Hold« und »Halt«. Wenn der Eingang aktiv ist, wird das Inkrementieren des Programmzählers (PC) gesperrt und $\overline{INTA}$ ausgegeben. Während dieses Zyklus kann ein RESTART- oder CALL-Befehl für den Sprung zu dem Interrupt-Bedienungsprogramm eingefügt werden. Der Eingang INTR wird über Software freigegeben und gesperrt. Er wird ferner durch »Reset« sowie unmittelbar nach Annahme einer Unterbrechungsanforderung gesperrt.
$\overline{INTA}$	11	A	INTERRUPT-Quittung: Der Ausgang wird in dem Befehlszyklus nach Annahme eines INTR-Signals anstelle von $\overline{RD}$ verwendet (und hat den gleichen Zeitablauf). Er kann zur Aktivierung des Unterbrechungs-Steuerbausteins SAB 8259A oder eines anderen Interrupt-Kanals verwendet werden.

Bild 3.1.2, Teil 2

Bezeichnung (Symbol)	Nummer	Eingang (E) Ausgang (A)	Funktion
RST 5.5 RST 6.5 RST 7.5	9 8 7	E	RESTART-Interrupt: Diese drei Eingänge haben den gleichen Zeitablauf wie INTR, bewirken jedoch, daß automatisch ein interner RESTART-Befehl eingefügt wird. Diese Interrupts haben eine höhere Priorität als INTR. Außerdem können sie durch Verwendung des Befehls SIM einzeln maskiert werden.
TRAP	6	E	Der Interrupt-Eingang TRAP ist ein nicht maskierbarer RESTART-Interrupt-Eingang. Er wird gleichzeitig mit INTR oder RST 5.5–7.5 abgefragt und von keiner Maske oder Interrupt-Freigabe beeinflußt. Er hat die höchste Priorität.
$\overline{\text{RESET IN}}$	36	E	Ein L-Signal an diesem Eingang setzt den Programmzähler auf 0 und das Interrupt-Freigabe- sowie HLDA-Flipflop zurück. Der Daten- und Adressen-Bus sowie die Steuerleitungen sind, solange RESET ansteht, hochohmig. Da RESET asynchron arbeitet, können die internen Register und Flags des Prozessors durch RESET mit undefinierbaren Ergebnissen verändert werden. $\overline{\text{RESET IN}}$ ist ein Schmitt-Trigger-Eingang und ermöglicht den Anschluß einer R-C-Kombination zur RESET-Verzögerung beim Zuschalten der Versorgungsspannung. Die Zentraleinheit bleibt so lange zurückgesetzt, wie $\overline{\text{RESET IN}}$ ansteht.
RESET OUT	3	A	Zeigt an, daß die Zentraleinheit rückgesetzt wird. Dieser Ausgang kann als System-Rücksetzsignal verwendet werden. Das Signal ist mit dem Prozessor-Takt synchronisiert. Seine Dauer erstreckt sich über eine ganze Zahl von Taktperioden.
X_1, X_2	1, 2	E	An X_1 und X_2 wird ein Quarz, eine LC- oder eine RC-Kombination zur Anregung des internen Taktgenerators angeschlossen. X_1 kann auch als Eingang für einen externen Takt verwendet werden. Die Eingangsfrequenz wird halbiert und ergibt die interne Arbeitsfrequenz des Prozessors.
CLK	37	A	Taktausgang zur Verwendung als Systemtakt. Die Periodendauer des Signals CLK beträgt das Doppelte von der an X_1, X_2.
SID	5	E	Eingang für serielle Datenübertragung. Sein Zustand wird bei jeder Ausführung eines RIM-Befehls in das Akku-Bit 7 übertragen.
SOD	4	A	Ausgang für serielle Datenübertragung. Der Ausgang SOD wird entsprechend der Form des Befehls SIM gesetzt bzw. rückgesetzt.
V_{cc}	40	–	Versorgungsspannung (+ 5 V)
V_{ss}	20	–	Masse (0V)

Bild 3.1.2, Teil 3

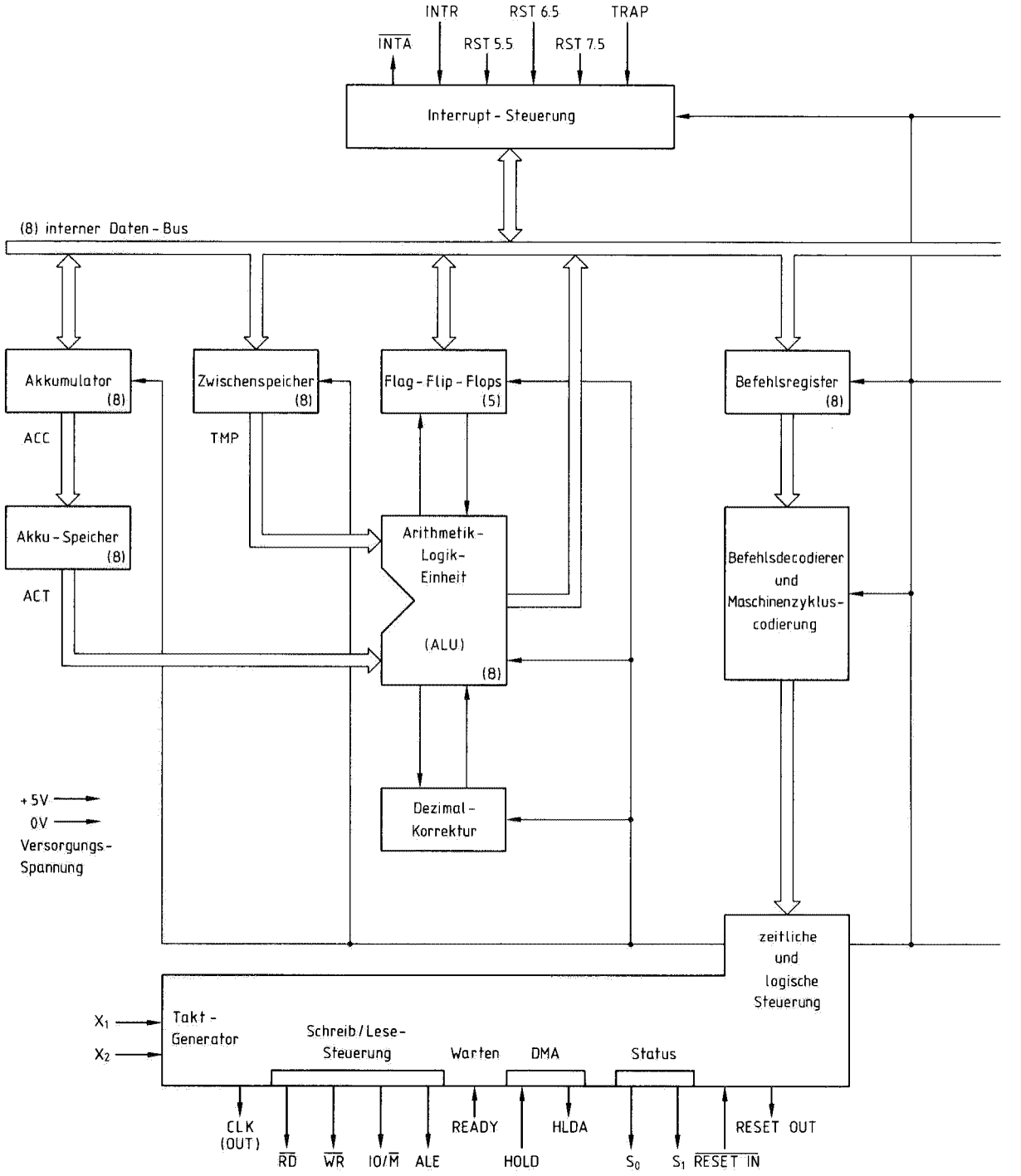
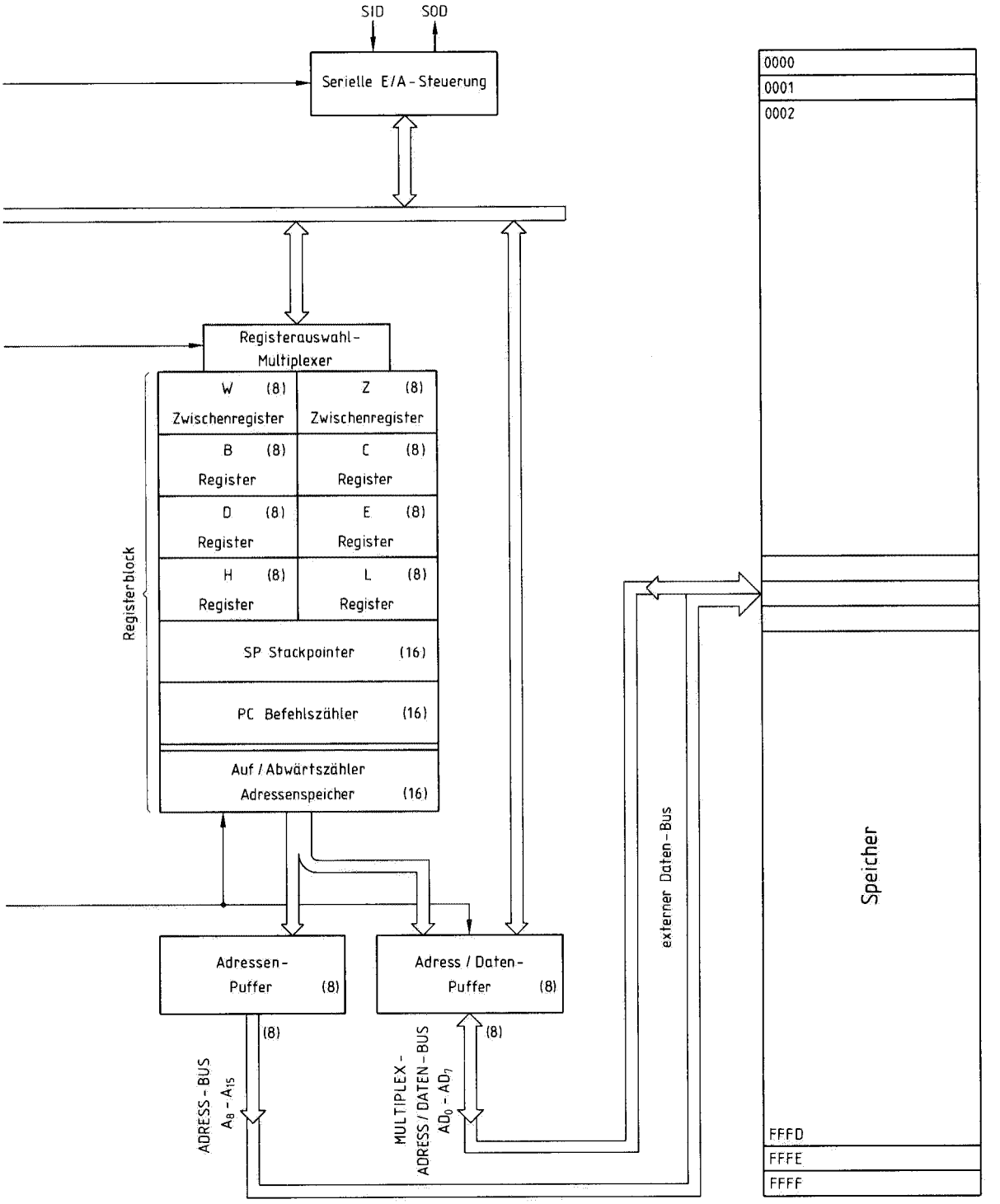


Bild 3.1.3: Das Funktions-Blockschaltbild des Prozessors 8085 A.


SID
SOD
Serielle E/A-Steuerung
Registerauswahl-Multiplexer
W (8) Zwischenregister
Z (8) Zwischenregister
B (8) Register
C (8) Register
D (8) Register
E (8) Register
H (8) Register
L (8) Register
SP Stackpointer (16)
PC Befehlszähler (16)
Auf / Abwärtszähler Adressenspeicher (16)
Registerblock
Adressen-Puffer (8)
Adress / Daten-Puffer (8)
(8)
(8)
ADRESS-BUS $A_8 - A_{15}$
MULTIPLEX-ADRESS / DATEN-BUS $AD_0 - AD_7$
externer Daten-Bus
0000
0001
0002
FFFD
FFFE
FFFF
Speicher

3.1 Der Mikroprozessor 8085 A

Der Mikroprozessor-Baustein 8085 A ist mit den Typen 8080 A und Z 80 insoweit software-kompatibel, daß alle für den 8085 bzw. 8085 A geschriebenen Programme auch in den Prozessoren 8080 A und Z 80 ablauffähig sind, sofern sie die erweiterten Interrupt-Eigenschaften des 8085 nicht nutzen. Die Eigenschaften bzw. Besonderheiten des 8085 A seien hier kurz aufgezählt:

- ca. 3 MHz Taktfrequenz, d. h. 1 Taktzyklus dauert rund 325 ns;
- kurze Befehlszyklus-Zeiten (ca. 1,3 bis 6 μs);
- es ist ein 3-Chip-Minimalsystem möglich, und zwar mit dem E/A-ROM-Baustein 8355 und dem E/A-Baustein 8155, der auch 1/4 K RAM und einen Zähler enthält;
- Multiplex-Betrieb auf dem Adreß-/Daten-Bus, d. h. das niederwertige Adreßbyte und das Datenbyte gehen über dieselben 8 Leitungen (vgl. hierzu *Kapitel 1*, Abschnitt „Die Bus-Struktur");
- einfache Schnittstelle zu Standardspeichern;
- einfache serielle Schnittstelle;
- priorisierte und maskierbare Interrupt-Eingänge;
- nur eine Versorgungsspannung (+5 V).

Den Anschlußplan des 40poligen Bausteins 8085 A zeigt *Bild 3.1.1*. Die Tabelle in *Bild 3.1.2* gibt nähere Erläuterungen zur Funktion der einzelnen Anschlüsse.

Der Aufbau des Mikroprozessors 8085 A

Das *Bild 3.1.3* zeigt den inneren Aufbau des 8085 A und seine prinzipielle Zusammenschaltung mit dem Speicher über den hier als extern bezeichneten Adreß-/Daten-Bus.

Neben einigen jeweils dazugehörenden Registern erkennt man in *Bild 3.1.3* folgende wichtige Bestandteile der CPU:

1. Arithmetik-Logik-Einheit (ALU)
2. Befehlsregister und Steuerlogik
3. Registerblock und Adressenlogik

Diese Teile der CPU sollen hier nun kurz besprochen werden. Die genaue Bedeutung und Funktion einzelner Register wird vor allem beim Durchführen der Experimente (ab *Seite 62*) verständlich werden. An dieser Stelle geht es um ein erstes Kennenlernen der CPU; auch zum Nachschlagen beim Durcharbeiten der nächsten Kapitel kann dieser Abschnitt hilfreich sein.

3.1.1 Arithmetik-Logik-Einheit

Sie wird auch kurz ALU genannt, von: Arithmetic Logic Unit. Die ALU besorgt die Durchführung von arithmetischen und logischen Operationen. Sie besteht aus einem Addier- und Subtrahiernetzwerk sowie logischen Verknüpfungsnetzwerken wie UND-, ODER- und Negationsgliedern sowie Äquivalenz- und Antivalenz-Schaltungen.
Die ALU benötigt zur Funktion zusätzlich folgende Register:
Akkumulator (8 Bit),
Akkumulator-Speicher (8 Bit),
Zwischenspeicher (8 Bit),
Zustandsregister (5 Bit).
Letzteres wird auch *Flag-Register* genannt. Es ist eigentlich ebenfalls ein 8-Bit-Register, jedoch sind 3 Bit bedeutungslos (vgl. *Kapitel 4.2*). Im Flag-Register vermerkt der Prozessor Besonderheiten des Ergebnisses einer Operation durch Setzen bzw. Rücksetzen der 5 Flip-Flops. Diese 5 Bits sind das Null-Bit, das Übertrags- oder Carry-Bit, das Vorzeichen-Bit, das Paritäts-Bit und das Hilfsübertrags-Bit (zur Dezimalkorrektur benötigt).
Die ALU bekommt ihre Daten von dem Zwischenspeicher, dem Akku-Speicher und dem Übertrags-Flip-Flop. Das 8-Bit-Ergebnis einer logischen oder arithmetischen Operation geht über den internen Datenbus in den Akkumulator. Besonderheiten des Ergebnisses werden – wie gesagt – im Flagregister gespeichert. Der Zwischenspeicher bekommt seine Information vom internen Datenbus und übermittelt sie an die ALU.
Der Akkumulator kann von der ALU über den internen Datenbus geladen werden. Der Akku überträgt Daten zum Akku-Speicher oder an den internen Datenbus. Der Akku-Inhalt und das Hilfsübertrags-Bit können während der Ausführung des DAA-Befehls auf Dezimalkorrektur hin überprüft werden. (Der DAA-Befehl gibt die Möglichkeit, neben Dual-Zahlen auch BCD-Zahlen zu verarbeiten.)

3.1.2 Befehlsregister und Steuerlogik – Der Ablauf von Befehlen

Die Funktion des Befehlsregisters und der Steuerlogik wird am besten anhand der Abläufe erklärt, die beim Abarbeiten eines Programmschrittes (Befehls) vor sich gehen. Dazu sei zunächst noch einmal daran erinnert (vgl. *Seite 12*), daß das erste Byte eines

Befehlswortes stets ein Op-Code (Befehls-Byte) sein muß.
Es laufen für jeden Befehl folgende vier Schritte ab:
1. Der Mikroprozessor – d. h. genauer: die Steuerlogik – sendet einen Aktivitätsbefehl auf den Steuerbus: Lesen einer Speicherzelle im ROM; dies kann, z. B. während der Testphase, auch im RAM geschehen; entscheidend ist allein die Adresse, siehe Punkt 2.
2. Die Steuerlogik sendet über den Adreßbus diejenige Adresse zum Speicher, die gerade im Befehlszähler steht (zum Befehlszähler siehe nächstes Teilkapitel).
3. Das im Speicher gelesene Byte gelangt über den Datenbus ins Befehlsregister.
4. Das jetzt im Befehlsregister befindliche Byte dient zur Steuerung der Abläufe während des Befehls (z. B. Einholen von Daten aus dem Speicher, logische oder arithmetische Operationen, usw.).
Nach der Decodierung des im Befehlsregister stehenden Op-Code-Bytes werden dazu ggf. auch entsprechende Ausgangsleitungen der Steuerlogik aktiviert (z. B.: Ansteuerung der E/A-Bausteine, Speicher-Freigabe für Lesen bzw. Schreiben, etc.).
Noch einige Sätze zur Steuerlogik: Sie weist auch Eingangsleitungen auf, über die sie (bzw. der Mikroprozessor) auf externe Signale reagieren kann, z. B. auf eine Interrupt-(Unterbrechungs-)Anforderung oder eine Warte-Anforderung.
Eine Unterbrechungs-Anforderung (Interrupt) veranlaßt die Steuerlogik, die Ausführung des Hauptprogramms zeitweilig zu unterbrechen und in ein Sonderprogramm zu springen. Dieses bedient das unterbrechende Gerät, das wichtige Daten weiterzugeben hat, und kehrt anschließend automatisch ins Hauptprogramm zurück.
Eine Warte-Anforderung wird oft von einem Speicher oder Ein-/Ausgabe-Baustein erteilt, der langsamer als der Prozessor arbeitet. Die Steuerlogik läßt dann den Prozessor stillstehen, bis der Speicher oder der E/A-Baustein seine Daten bereit hat.

Der genaue Ablauf eines Befehlszyklus

Der an den Pins 1 und 2 des Prozessor-Bausteins angeschlossene Schwingquarz-Taktgeber ordnet die Arbeit der CPU zeitlich; er hat im ECB 85 eine Frequenz von 6,144 MHz. Im Prozessor 8085 A wird die Eingangs-Frequenz zunächst auf die Hälfte heruntergeteilt, so daß ein einzelner Takt bzw. *Taktzyklus* ca. 325 ns dauert. Diese Taktzeit ist also die kleinstmögliche Zeiteinheit, in die man die beim Programmablauf auftretenden Zeitspannen unterteilen kann. In der Befehlsliste *(Seite 140ff.)* ist angegeben, wieviele Taktzyklen das Einholen und Ausführen des jeweiligen Befehls in Anspruch nimmt. Diese Zeitspanne nennt man *Befehlszyklus*. Jeder Befehlszyklus besteht also stets aus mehreren Taktzyklen. Ein weiterer wichtiger Begriff in diesem Zusammenhang ist der *Maschinenzyklus*. Vereinfacht gesagt, gilt: Ein neuer Maschinenzyklus beginnt während der Ausführung eines Befehls immer dann, wenn der Prozessor in einen neuen Datenaustausch mit dem Speicher oder einem E/A-Baustein treten muß.
Das *Bild 3.1.4* macht deutlich, daß die beim Ablauf eines Programms aufeinanderfolgenden Befehlszyklen jeweils aus (meist) mehreren Maschinenzyklen bestehen; jeder Maschinenzyklus selbst umfaßt seinerseits stets mehrere Taktzyklen. Der erste Maschinenzyklus eines Befehls ist beim 8085 A immer 6 oder 4 Taktzyklen lang, und jeder weitere dann 3 Taktzyklen.

Beispiel für einen Befehlsablauf: Es wird das folgende Beispielprogramm betrachtet *(Seite 20 oben)*.

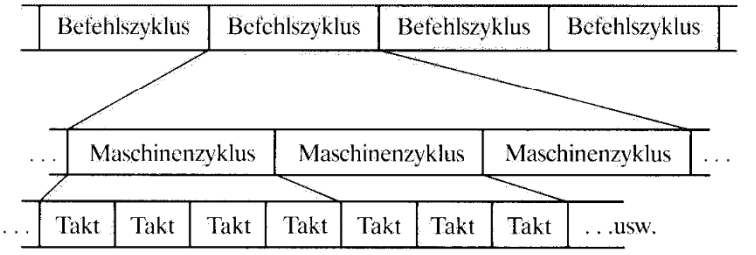


Bild 3.1.4: Der Zusammenhang zwischen Befehls-, Maschinen- und Taktzyklus. Jeder Befehlszyklus besteht aus mindestens einem Maschinenzyklus und dieser stets aus mehreren Taktzyklen von jeweils 325 ns Dauer (vgl. Text).

Adresse	Hex-Code	Mnemo-Code
19A7	3E	MVI A, 0EH
19A8	0E	
19A9	D3	OUT F9
19AA	F9	

Zunächst wird also der Akkumulator mit 0EH geladen (MVI A, 0EH); dieser Wert soll dann auf die E/A-Adresse F9 (vgl. dazu *Kapitel 3.3*) ausgegeben werden. Beim Ablauf dieses Befehls OUT F9 laufen drei Maschinenzyklen ab (vgl. *Bild 3.1.5*).

1. Maschinenzyklus (M1, bestehend aus 4 Taktzyklen T1 bis T4)
1. Takt: Zunächst wird auf dem Steuerbus das Signal ALE auf 1 gesetzt (Adressenspeicher-Freigabe) und IO/$\overline{M}$ auf 0; es soll also aus dem Speicher gelesen werden. Dann wird die Adresse, d. h. der Inhalt des Befehlszählers, auf den Adreßbus gegeben, und zwar das höherwertige Byte (19) auf die Leitungen $A_8 - A_{15}$ und das niederwertige (A9) auf $AD_0 - AD_7$ (gemultiplexter Adreß-/Daten-Bus). Der Befehlszähler muß zu Beginn natürlich eine Speicheradresse enthalten, unter der ein Op-Code-Byte (Befehlsbyte) abgelegt wurde. (Der Befehlszähler-Inhalt wird inkrementiert, d. h. um 1 erhöht; dies ist im Diagramm nicht gezeigt.)

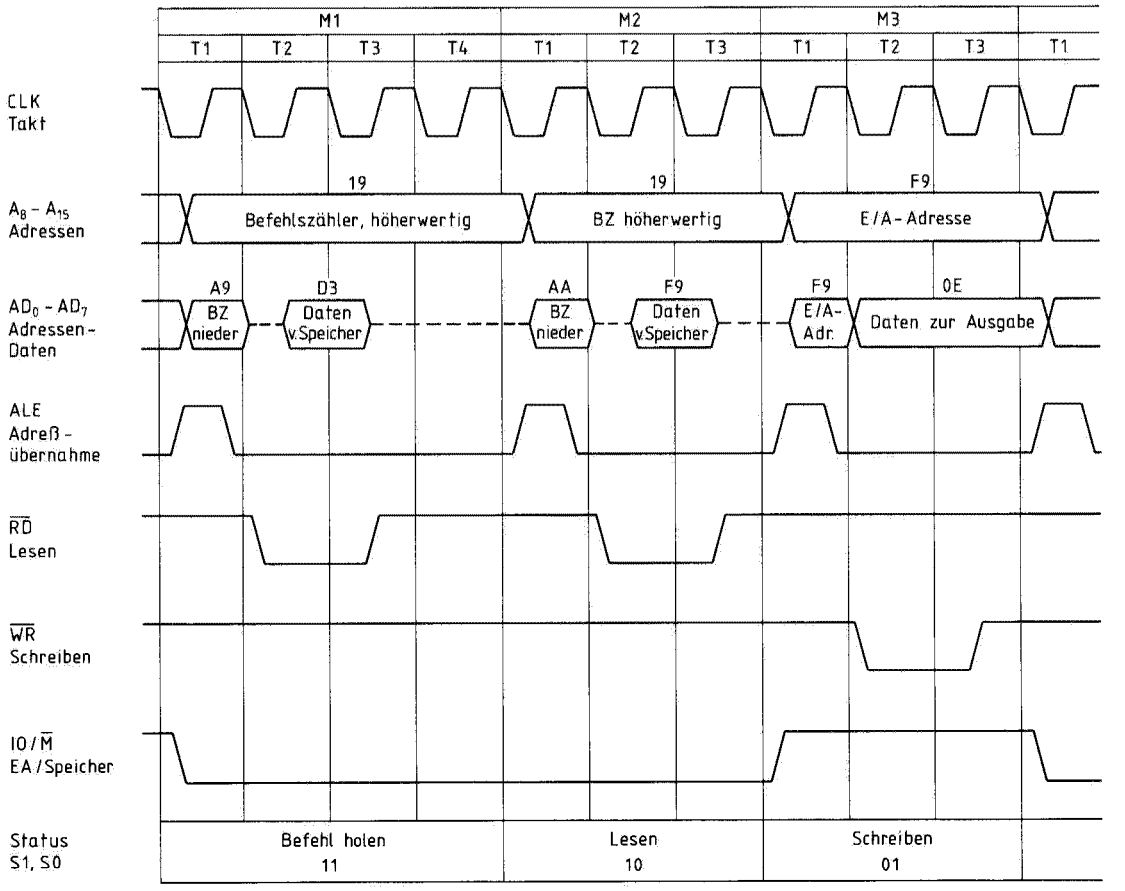


Bild 3.1.5: Die Abläufe beim Ausführen des Befehls OUT F9H (im Hex.-Code: D3 F9), wenn zuvor der Akkumulator der CPU mit 0E H geladen wurde und das erste Byte des OUT-Befehls in der Speicheradresse 19A9 steht.

2. und 3. Takt: Die Steuerleitung $\overline{RD}$ wird auf 0 gesetzt: Es wird nur gelesen (engl. read, daher RD), und der Adreß-/Daten-Bus steht für die Datenübertragung vom Speicher zum Prozessor zur Verfügung; er übernimmt jetzt das „Daten"-(eigentlich Op-Code-)Byte D3.
4. Takt: (Der Befehl würde im 4. Takt fertig abgearbeitet, wenn er nur aus einem Maschinenzyklus bestünde, was z. B. bei MOV A,B der Fall wäre). Der Prozessor interpretiert das gelesene Byte D3 als Op-Code (da es das erste Byte eines Befehls ist), speichert es im Befehlsregister und decodiert es.

2. Maschinenzyklus (M2, T1 bis T3)
Entsprechend dem gelesenen Befehlsbyte (D3 ≙ OUT) soll das vom vorigen Befehl noch im Akkumulator stehende Datenbyte (0E) an diejenige Adresse ausgegeben werden, die im Programmspeicher auf den Op-Code folgt, d. h. die in der Speicheradresse 19AA steht. Es muß also der nächste Speicherplatz gelesen werden; seine Adresse steht schon im (vorher inkrementierten) Befehlszähler: 19AA. In den 3 Takten dieses 2. Maschinen-Zyklus geschieht nun genau dasselbe wie im 1. Maschinen-Zyklus: ALE auf 1 und Aussenden der im Befehlszähler stehenden Adresse; dann $\overline{RD}$ auf 0 und dabei Übernahme des Datenbytes (F9) aus dem Speicher auf den Datenbus.

3. Maschinenzyklus (M3, T1 bis T3)
Hier erfolgt die Ausgabe des Akku-Inhalts (0E) auf die Adresse F9 wie folgt:
1. Takt: ALE auf 1 (Adreß-Freigabe) und IO/$\overline{M}$ auf 1, d. h. ein E/A-Baustein (IO, von engl. in, out) soll angesprochen werden. Weiterhin liegt $\overline{RD}$ auf 1, und das höherwertige Adreß-Byte A_8-A_{15} übernimmt die Information (F9) wie die Leitungen AD_0-AD_7 als E/A-Adresse.
2. und 3. Takt: $\overline{WR}$ auf 0, d. h. es soll auf die gerade angewählte Speicher- oder E/A-Adresse ausgegeben werden („geschrieben", engl. write, daher WR); da aber IO/$\overline{M}$ auf 1 liegt, werden die im Akku befindlichen Daten (0E) auf eine E/A-Adresse (F9) ausgegeben.
Der nächste Befehl beginnt wieder, wie am rechten Rand der Abbildung angedeutet, mit derselben Wertekombination wie der gerade betrachtete: ALE auf 1 und IO/$\overline{M}$ auf 0, d. h. das nächste (Befehls-) Byte wird aus dem Speicher geholt.

3.1.3 Registerblock und Adressenpuffer

Im Adreß-/Daten- und Adressen-Puffer werden Daten bzw. Adressen (AD_0-AD_7 und A_8-A_{15}) zwischengespeichert, so daß sie am Adreß-/Daten-Bus so lange wie erforderlich zur Verfügung stehen.
Etwas eingehender müssen die im Registerblock zusammengefaßten *Register* (statische RAMs) des Mikroprozessors 8085 A betrachtet werden. Die Kenntnis ihrer Funktionen und Eigenschaften ist für den Programmierer sehr wichtig. Die Auswahl der gerade anzusprechenden Register besorgt der Registerauswahl-Multiplexer. Über ihn werden 8-Bit-Datenwörter zwischen dem internen Datenbus und der Register-Matrix (dem Registerblock) übertragen.
Der Registerblock enthält (siehe *Bild 3.1.3*):
Vier Paar 8-Bit-Register:
W, Z
B, C
D, E
H, L
Zwei 16-Bit-Register:
SP (Stackpointer, Stapelzeiger)
PC (Program Counter, Befehlszähler)
Einen 16-Bit-Adressenspeicher mit Auf-/Abwärts-Zähler.
Über ihn können 16-Bit-Datenwörter übertragen werden, die er von einem der Registerpaare WZ, BC, DE und HL erhält.

Die Beschreibung der Register im einzelnen:

W und Z:
Diese beiden 8-Bit-Zwischenregister sind dem Programmierer nicht zugänglich; sie werden hier nicht weiter betrachtet.
B und C
D und E
H und L:
Diese sechs allgemeinen 8-Bit-Register können durch Programmbefehle entweder jeweils einzeln angesprochen werden, oder aber sie sind als drei 16-Bit-Register anzusehen, wobei jeweils ein Paar zusammengefaßt wird:
BC unter der Bezeichnung B,
DE unter der Bezeichnung D,
HL unter der Bezeichnung H.
Eine Sonderrolle spielt das Registerpaar HL bei der sogen. indirekten Adressierung (siehe *Kapitel 4.1*).

SP (Stackpointer):
Er enthält stets die Adresse, unter der zuletzt im Kellerspeicher (engl. stack) etwas abgelegt wurde; vgl. hierzu den Aufruf von Unterprogrammen, *Seite 41*.
PC (Program Counter):
Er enthält die Adresse des als nächstes aufzurufenden Befehlsbytes; vgl. hierzu die Beschreibung des Signalverlaufs beim Abarbeiten eines Befehls im vorigen Abschnitt.

3.2 Der Speicher im ECB 85

Es war in den vorangegangenen Abschnitten schon mehrmals die Rede von Speicheradressen, unter denen Befehls- oder Datenbytes abgelegt sind. Des weiteren wurden in Kapitel 1 schon einige allgemeine Ausführungen über Speicher gemacht. Nun soll

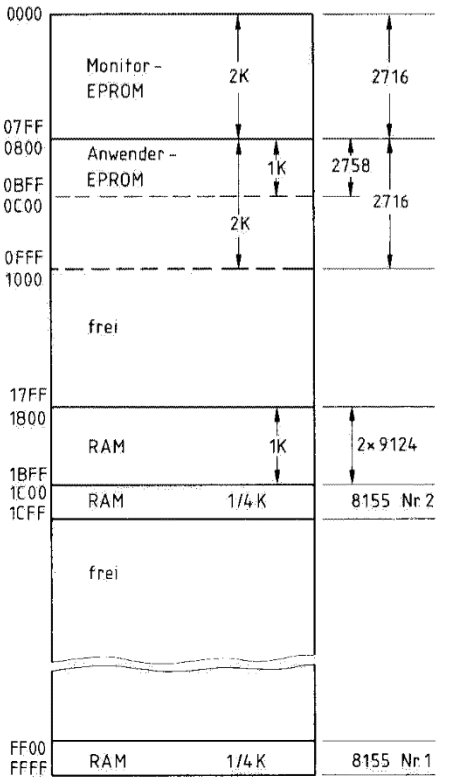


Bild 3.2.1: Der Speicher-Adreßraum des ECB 85 mit hexadezimaler Angabe der Adresse (links), der Kapazität der einzelnen Bereiche und der Bausteintypen (rechts).

der Aufbau des Speichers im ECB 85 erläutert werden. Der Mikroprozessor 8085 A kann – wie ebenfalls schon erwähnt – mit seinem 16-Bit-Adreßbus einen Speicher der Kapazität 64 K ($=2^{16}=65536$ Bytes) verwalten. Berücksichtigt man, daß die Speicherplatz-Numerierung bei Null beginnt, so ist die größtmögliche Speicherplatz-Adresse 65535. Rechnet man diese Dezimalzahl in das Hexadezimal-System um (vgl. *Bild 2.1*), so ergibt sich FFFF ($=15*16^3+15*16^2+15*16^1+15*16^0$).
Mit anderen Worten: Der sogenannte *Adreßraum* im Speicher des ECB 85 umfaßt die Adressen von 0000 bis FFFF. Dies ist in *Bild 3.2.1* dargestellt. Diese Abbildung zeigt auch die Aufteilung des insgesamt möglichen Adreßraums in verschiedene Bereiche unterschiedlicher Funktion. Wie hier aus Umfangsgründen nicht näher erläutert werden kann, muß diese Aufteilung z. B. in RAM- und EPROM-Bereiche natürlich nicht bei jedem Mikrocomputer mit dem Prozessor 8085 A genauso aussehen; jedoch muß das Betriebs- oder Monitor-Programm (s. u.) stets bei der Adresse 0000 beginnen.
Man erkennt im wesentlichen drei Speicherbereiche:
1. Das Monitor-EPROM
(Baustein 2716, mit 2 K, Adr. 0000 – 07FF)
2. Das Anwender-EPROM
(Baustein 2758, mit 1 K, Adr. 0800 – 0BFF,
oder
Baustein 2716, mit 2 K, Adr. 0800 – 0FFF)
3. Drei RAM-Bereiche
(2 Bausteine 9124, mit 1 K, Adr. 1800 – 1BFF,
1 Baustein 8155, mit 1/4 K, Adr. FF00 – FFFF,
und evtl.
1 Baustein 8155, mit 1/4 K, Adr. 1C00 – 1CFF;
dieser letztgenannte Bereich kann auch frei sein, wenn der Sockel für den Baustein 8155 Nr. 2 leer ist, vgl. *Kapitel 3.3*).
Zu den genannten Bereichen sind einige Erläuterungen angebracht:
Das *Monitor-Programm* ermöglicht überhaupt erst die Kommunikation des Benutzers mit dem Mikroprozessor über Tastatur und Anzeigefeld, und zwar ohne zusätzliche Geräte. Das Monitor-Programm (das Wort Monitor hat hier nichts mit Bildschirm zu tun) dient dazu, den Betrieb des Mikrocomputers zu steuern und zu überwachen; dazu fragt es u. a. die Tastatur ab, verarbeitet die hier eingegebenen Signale, prüft sie auf Zulässigkeit, gibt am Anzeigefeld Meldungen aus, usw. Für die Funktion des Monitor-Programms (Adr. 0000 – 07FF) ist außerdem noch der RAM-Bereich zwischen den Adressen

FFB3 und FFFF notwendig; er darf vom Anwender nicht benutzt werden.
Das *Anwender-EPROM* (z. B. Typ 2716) kann auf dem freien Steckplatz beim Tastenfeld eingesetzt werden, wenn mit fehlerfreien und nicht mehr zu ändernden Programmen oder Daten gearbeitet werden soll. Auf diesem Sockel kann auch ein zuvor gelöschtes EPROM eingesetzt werden, das dann z. B. mit dem Kommando 2 programmiert werden kann (siehe *Kapitel 6*).
Der *RAM-Bereich* schließlich wird beim Experimentieren am häufigsten verwendet. Er besteht aus zwei Bausteinen vom Typ 9124, mit denen der Adreßbereich von 1800 bis 1BFF realisiert wird (1 K). Dazu kommen noch – je nach Bestückung des ECB 85 – ein oder zwei 1/4-K-Bereiche, wie in *Bild 3.2.1* angegeben. Sie werden durch ein oder zwei Bausteine des Typs 8155 realisiert. Mit den wesentlich wichtigeren E/A-Funktionen dieser Bausteine befaßt sich der nächste Abschnitt.

3.3 Die Ein-/Ausgabe-Bausteine 8155

Wenn der Mikrocomputer mit externen („peripheren") Geräten oder Experimentierschaltungen verbunden werden soll, so muß er neben Prozessor und Speicher noch über eine Eingabe- und/oder Ausgabe-Einrichtung verfügen; vgl. *Bild 1.2*. Der ECB 85 besitzt dazu einen Baustein des Typs 8155; ein zweites 8155-IC kann auf dem dafür in der Platine fertig verdrahteten freien Sockel nachgerüstet werden. Der Speicher-Teil des 8155 ist in *Kapitel 3.2* schon besprochen worden.
Im folgenden werden der Timer und die E/A-Ports des Bausteins 8155 eingehend behandelt; er ist mit dem 8156 identisch – abgesehen vom Pegel des CE-Signals zur Baustein-Auswahl (s. u.), das beim 8156 aktiv high (CE) und beim 8155 aktiv low ($\overline{CE}$) ist.
Die Anschlüsse des E/A-Bausteins 8155 sind für den Einsatz am Prozessor 8085 bzw. 8085 A angepaßt. Er kann also die gemultiplexten Adreß/Daten-Signale AD_0–AD_7 mit Hilfe des ALE-Signals direkt verarbeiten. Der 8155 besitzt einen Zwischenspeicher („Adreß-Latch"), in dem mit der fallenden Flanke des ALE-Signals die Speicher- oder E/A-Adresse (s. u.) gespeichert wird.
Neben den 8 Signalen AD_0–AD_7 des Adreß/Daten-Bus werden vom Prozessor-Baustein die Steuersignale ALE (schon erwähnt), IO/$\overline{M}$, $\overline{RD}$ und $\overline{WR}$ auf den 8155 geführt. Zur Bedeutung dieser Steuersignale vgl. auch noch einmal *Bild 3.1.1* und *3.1.2*.
Die Signale ALE und IO/$\overline{M}$ werden gemeinsam mit AD_0–AD_7 im schon erwähnten Zwischenspeicher des 8155 abgelegt. Dieser empfängt auch das Bausteinauswahl-Signal $\overline{CE}$, das low-Pegel haben muß, damit der Baustein 8155 angesprochen werden kann. Das $\overline{CE}$-Signal kommt nicht direkt vom Prozessor, sondern wird in einem eigenen Codierer erzeugt, dessen Eingänge mit dem Adreßbyte A_8–A_{15} und dem Steuersignal IO/$\overline{M}$ verbunden sind. Dabei sei daran erinnert, daß z. B. beim Abarbeiten eines OUT-Befehls (siehe *Bild 3.1.5*) die E/A-Adresse (s. u.) auch auf das höherwertige Adreßbyte A_8–A_{15} gegeben wird.

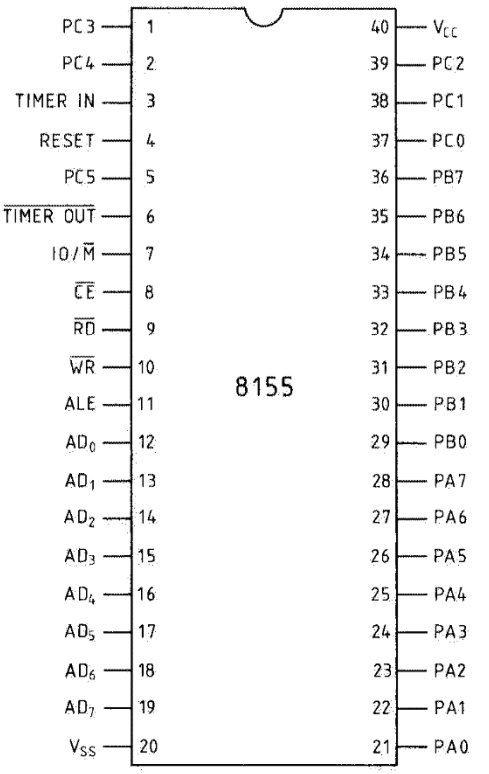


Bild 3.3.1: Anschlußplan des E/A-Bausteins 8155 bzw. des 8156 (bei diesem ist das Signal CE statt $\overline{CE}$ auf Pin 8 zu legen). Die Ausgänge bzw. Eingänge der Ports A, B und C sind hier mit PA0–PA7, PB0–PB7 und PC0–PC5 bezeichnet. Bei der Auflistung aller Signale, die auf die Anwender-Schnittstelle des ECB 85 geführt werden *(Bild 3.4.1)*, ist zur Unterscheidung noch die Nr. des 8155-Bausteins angegeben, z. B.: P1A0 für den Anschluß PA0 des 8155 Nr. 1.

Bezeichnung	Pin Nr.	Eingang oder Ausgang	Funktion
RESET	4	E	Rücksetz-Eingang; er ist mit dem Signal RESET OUT des 8085 zu verbinden. Ein H-Signal setzt den 8155 zurück und initialisiert die drei E/A-Kanäle für die Eingabe.
$AD_0 - AD_7$	12 – 19	E/A	gemultiplexter Adreß/Daten-Bus, bidirektional, d.h. die Übertragung ist in beiden Richtungen möglich.
CE oder $\overline{CE}$	8	E	Bausteinauswahl-Signal. Es ist CE = 1 für den 8156 und $\overline{CE}$ = 0 für den 8155. Dieser aktive Zustand muß für die Wirksamkeit aller nachfolgend aufgeführten Signale vorliegen.
$\overline{RD}$	9	E	$\overline{RD}$ = 0: Lesen; es werden Daten aus dem RAM-Teil des 8155 oder von seinen E/A-Ports zur CPU übertragen.
$\overline{WR}$	10	E	$\overline{WR}$= 0: Schreiben; es werden Daten aus der CPU zum 8155 übertragen.
ALE	11	E	Steuersignal des 8085; ist ALE = 1, so werden auf dem gemultiplexten Adreß/Daten-Bus Adressen übertragen.
IO/$\overline{M}$	7	E	Steuersignal des 8085; mit IO/$\overline{M}$ = 1 werden die E/A-Ports des 8155 angesprochen, dagegen mit IO/$\overline{M}$ = 0 der Speicher, bei $\overline{CE}$ = 0 also der RAM-Teil des 8155. (Gleichzeitig wird mit $\overline{RD}$ = 0 bzw. $\overline{WR}$ = 0 festgelegt, ob gelesen bzw. geschrieben wird).
$PA_0 - PA_7$	21 – 28	E/A	Ein-/Ausgabe-Anschlüsse des Ports A; ob Eingabe oder Ausgabe vorliegt, wird durch das Steuerwort festgelegt.
$PB_0 - PB_7$	29 – 36	E/A	Ein-/Ausgabe-Anschlüsse des Ports B; vgl. die Bemerkung bei Port A.
$PC_0 - PC_5$	37 – 39 und 1, 2, 5	E/A	Ein-/Ausgabe-Anschlüsse des Ports C; vgl. die Bemerkung bei Port A. Bei Betrieb mit Rückmeldung führen die Anschlüsse des Ports C Steuer-Signale.
TIMER IN	3	E	Zähler- / Zeitgeber-Eingang
$\overline{TIMER\ OUT}$	6	A	Hier kann je nach Programmierung des Timers ein Rechtecksignal oder ein einzelner Impuls abgegriffen werden.
V_{cc}	40	–	Versorgungsspannung +5 V
V_{ss}	20	–	Masse 0 V

Bild 3.3.2: Die Anschlüsse des E/A-Bausteins 8155 bzw. 8156 und ihre Funktion.

Der Aufbau des E/A-Bausteins 8155

Nachdem soeben einige Vorbemerkungen über die Ansteuerung des 8155 gemacht wurden, sollen nun Aufbau und Funktionsweise dieses vielseitigen Bausteins ausführlich behandelt werden. *Bild 3.3.1* zeigt seinen Anschlußplan, und in *Bild 3.3.2* wird die Bedeutung der Anschlüsse angegeben.
Zum besseren Verständnis der Zusammenhänge der Ausführungen sei *Bild 3.3.3* näher betrachtet. In dieser Abbildung sind zu erkennen:

- der Zwischenspeicher mit den Eingängen $AD_0 - AD_7$ sowie $\overline{CE}$, $IO/\overline{M}$ und ALE;
- die Steuereingänge $\overline{RD}$ und $\overline{WR}$;
- der RAM-Speicher mit 1/4 K Byte, d. h. 256*8 Bit, siehe *Kapitel 3.2;* er interessiert an dieser Stelle nicht mehr;
- der Timer, auch Zähler genannt: dies ist ein programmierbarer 14-Bit-Abwärtszähler; er wird im letzten Abschnitt dieses Kapitels besprochen;
- die E/A-Kanäle, auch als E/A-Ports bezeichnet, mit jeweils einem Register, das die E/A-Leitungen

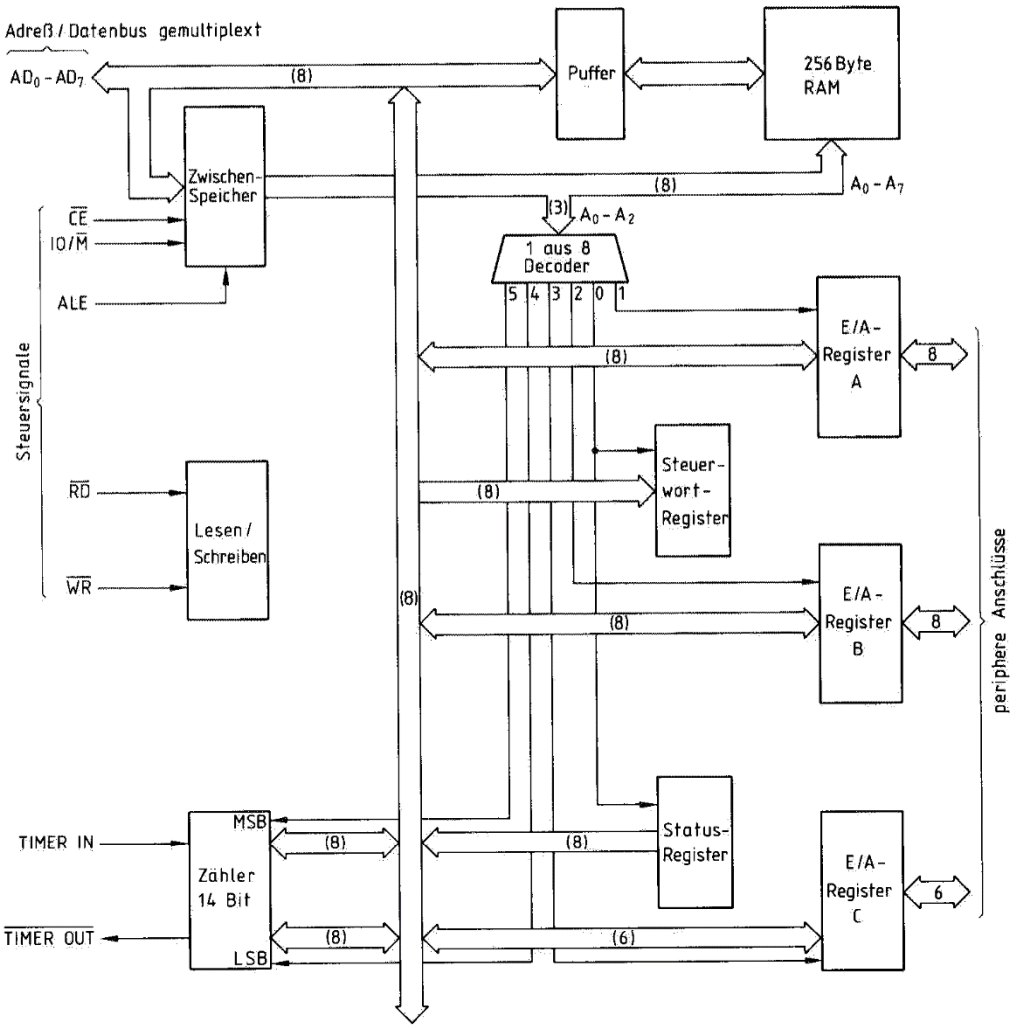


Bild 3.3.3: Das Blockschaltbild des E/A-Bausteins 8155. Ausführliche Erläuterung im Text.

steuert; Port A und B haben je 8 Bit, und Port C hat 6 Bit. Jeder der drei Ports ist als Eingabe- oder als Ausgabe-Port zu programmieren, s. u.;

- das Steuerwort-Register: es kann nur beschrieben, dagegen nicht gelesen werden;
- das Status-Register: es kann nur gelesen werden und ist unter derselben E/A-Adresse wie das Steuerwort-Register anzusprechen.

Die E/A-Adressen

Wie schon erwähnt, wird durch die vom Prozessor abgegebenen Steuersignale ALE, IO/$\overline{M}$, $\overline{RD}$ und $\overline{WR}$ sowie durch $\overline{CE}$ festgelegt, wann ein bestimmter E/A-Baustein zum Schreiben bzw. Lesen bestimmter Register angesprochen wird; dies geschieht z. B. durch die Prozessor-Befehle IN bzw. OUT (s. u.). Wird der E/A-Baustein angesteuert, dann wird das auf den Adreßbus AD_0-AD_7 gegebene Byte nicht als niederwertiges Byte einer Speicher-Adresse interpretiert, sondern als einziges Byte einer sogen. Kanal-Nummer, auch E/A-Adresse genannt.

Für die Adressierung der Register im Baustein 8155 werden, wie *Bild 3.3.3* zeigt, die drei niedrigstwertigen Bits A_0-A_2 der Adresse auf einen 1-aus-8-Decoder geführt. Dessen 6 Ausgangsleitungen (0 bis 5) dienen zur Ansteuerung der Register.
Zunächst seien Steuerwort- und Status-Register betrachtet, die ja (wie schon erwähnt) dieselbe Adresse haben. Zu ihrer Ansteuerung dient der Codierer-Ausgang 0. Damit sie aktiviert wird, müssen also die drei Adreßbits A_0-A_2 den Wert Null repräsentieren, d. h. alle 0-Wert führen. Das komplette Adreßbyte für das Steuerwort- und das Status-Register lautet also xxxxx000, wobei die 5 höherwertigen, hier mit x bezeichneten Bits für die Ansteuerung der Register im 8155 selbst belanglos sind (nur A_0-A_2 gehen ja auf den Decoder), jedoch u. a. durch das $\overline{CE}$-Signal festgelegt werden. Die vier höchstwertigen Bits sind im ECB 85 alle mit 1-Wert anzusetzen, und das 2^3-Bit ist 1 beim 8155 Nr. 1, dagegen 0 beim 8155 Nr. 2. Es resultieren für das Steuerwort- und das Status-Register also folgende Adressen:
11111000 B = F8 H für den 8155 Nr. 1,
und: 11110000 B = F0 H für den 8155 Nr. 2.

Decoder-ausgang Nr.	Register bzw. Port	Adresse				
		allgemein	für 8155 Nr. 1		für 8155 Nr. 2	
			bin.	hex.	bin.	hex.
0	Steuerwort- u. Status-R.	xxxxx000	11111000	F8	11110000	F0
1	Reg. A (Port A)	xxxxx001	11111001	F9	11110001	F1
2	Reg. B (Port B)	xxxxx010	11111010	FA	11110010	F2
3	Reg. C (Port C)	xxxxx011	11111011	FB	11110011	F3
4	Timer, LSB	xxxxx100	11111100	FC	11110100	F4
5	Timer, MSB	xxxxx101	11111101	FD	11110101	F5

Bild 3.3.4: Die Adressen der Ports und Register in den beiden 8155-Bausteinen des ECB 85. Die Bezeichnungen LSB und MSB beim Timer (der weiter unten behandelt wird) bedeuten:
LSB = Least Significant Byte = niedrigstwertiges Byte, und: MSB = Most Significant Byte = höchstwertiges Byte.

Ebenso wie für Steuerwort- und Status-Register kann aus *Bild 3.3.3* leicht entnommen werden, wie die drei niederwertigen Bits der Adressen der anderen Register im 8155 aussehen müssen. Man betrachtet also die Ausgänge des 1-aus-8-Decoders: Leitung 1 geht zum Port A, daher müssen die drei niederwertigen Adreßbits hier den Wert 1 repräsentieren (binär 001), und seine Adresse ist xxxxx001. Entsprechend verfolgt man alle weiteren Ausgänge des Codierers und erhält die in *Bild 3.3.4* aufgeführten Adressen, wobei die 4 höherwertigen Bits gleich 1 gesetzt werden. Das 2^3-Bit unterscheidet die beiden 8155-Bausteine.
Mit den hier erarbeiteten E/A-Adressen aller Register der beiden 8155-Bausteine des ECB 85 kann nun analog zum Adreßraum des Speichers der Adreßraum für die E/A-Funktionen übersichtlich

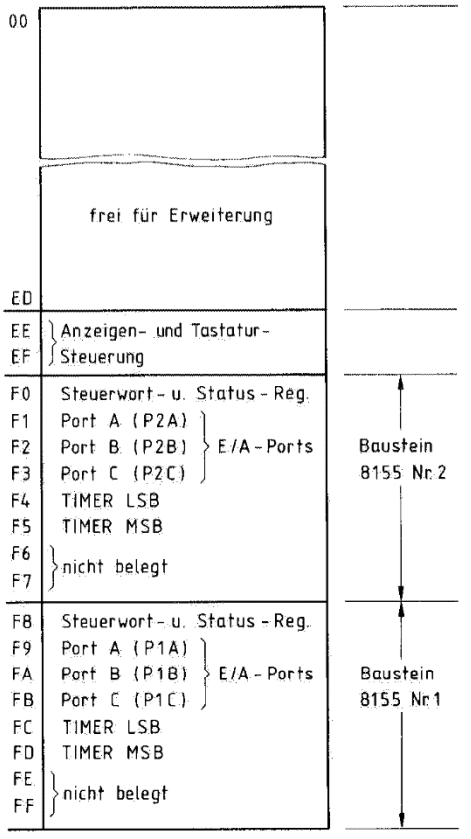


Bild 3.3.5: Der Ein-/Ausgabe-Adreßraum des ECB 85. Vgl. hierzu *Bild 3.3.4.*

dargestellt werden: *Bild 3.3.5.* In diesem Bild sind auch die beiden Adressen EE und EF für den Baustein 8279 angegeben, der für die Kommunikation mit Tastatur und Anzeigefeld sorgt; dies soll hier jedoch nicht weiter behandelt werden.

Die Initialisierung

Sollen in einem Programm für den 8085 die E/A-Ports oder der Timer eines 8155-Bausteins verwendet werden, so muß dieser zunächst initialisiert werden. In dieser sogen. Initialisierung ist die Betriebsart der E/A-Kanäle festzulegen, d. h. ihre Funktion als Eingabe- oder als Ausgabe-Port. Desgleichen muß, falls erforderlich, die Betriebsart des Timers festgelegt werden. Die Initialisierung geschieht dadurch, daß ein 1-Byte-Steuerwort (auch Kommando-Wort genannt) in das Steuerwort-Register des 8155 eingeschrieben wird. Soll der Timer benutzt werden, so ist er vorher mit 2 Bytes zu laden; Näheres dazu später.
Der Ablauf der Initialisierung: Zuerst muß das benötigte Steuerwort (s. u.) in den Akku des Prozessors gebracht werden, z. B. durch den Befehl MVI A,0DH, wobei das Steuerwort hier 0D (im Hex-Code) ist. (Wie das Steuerwort jeweils aussehen muß und welche Bedeutung es hat, ist Gegenstand der nächsten beiden Abschnitte). Dann ist dieses Steuerwort in das Steuerwort-Register des 8155 einzuschreiben, und zwar mit dem OUT-Befehl des Prozessors, z. B. OUT F8. Beim Ausführen des OUT-Befehls werden – wie schon angedeutet – von der CPU, d. h. vom Prozessor, die Signale IO/$\overline{M}$ = 1 und $\overline{WR}$ = 0 generiert (erzeugt); ebenfalls resultiert das zutreffende $\overline{CE}$-Signal, s. o.
Die nachfolgende Behandlung der E/A-Ports und des Timers bezieht sich – wenn nichts anderes gesagt wird – stets auf den Baustein 8155 Nr. 1 mit dem Adreßbereich F8–FD. Soll der Baustein 8155 Nr. 2 benutzt werden, so sind entsprechend die Adressen F0–F5 zu verwenden, jede angegebene Adresse also um 8 zu verringern (vgl. *Bild 3.3.5*).

Die Programmierung der E/A-Ports

Die Programmierung der E/A-Ports des 8155-Bausteins ist Bestandteil der oben schon beschriebenen Initialisierung: Es wird ein Steuerwort vom Akku der CPU in das Steuerwort-Register des 8155 eingelesen. Dieses Steuerwort legt also die Betriebsart der E/A-Ports fest.

Für jeden der Ports A, B und C sind folgende vier Grundbetriebsarten möglich:

1. Eingabe ohne Rückmeldung: Es werden durch einen IN-Befehl die Daten eingelesen (in den Akku der CPU gebracht), die gerade an den Port-Anschlüssen anliegen; sie dürfen sich während des Einlesevorgangs nicht ändern und werden auch nicht im E/A-Register (A, B oder C) zwischengespeichert, sondern beim Ausführen des IN-Befehls (vgl. dazu *Kapitel 5)* über den Datenbus in den Akku übertragen.

2. Eingabe mit Rückmeldung: soll hier nicht behandelt werden.

3. Ausgabe ohne Rückmeldung: Das betreffende E/A-Register (z. B. A) übernimmt mit einem OUT-Befehl (z. B. OUT F9) das Bitmuster des Akkumulators und speichert es, bis es durch einen neuen OUT-Befehl verändert wird; gleichzeitig wird das Bitmuster an die Peripherie-Anschlüsse des betreffenden Ports (z. B. A_0–A_7 des Bausteins 8155 Nr. 1, also die Peripherie-Anschlüsse P1A0 – P1A7 vgl. *Bild 3.4.1*) ausgegeben.

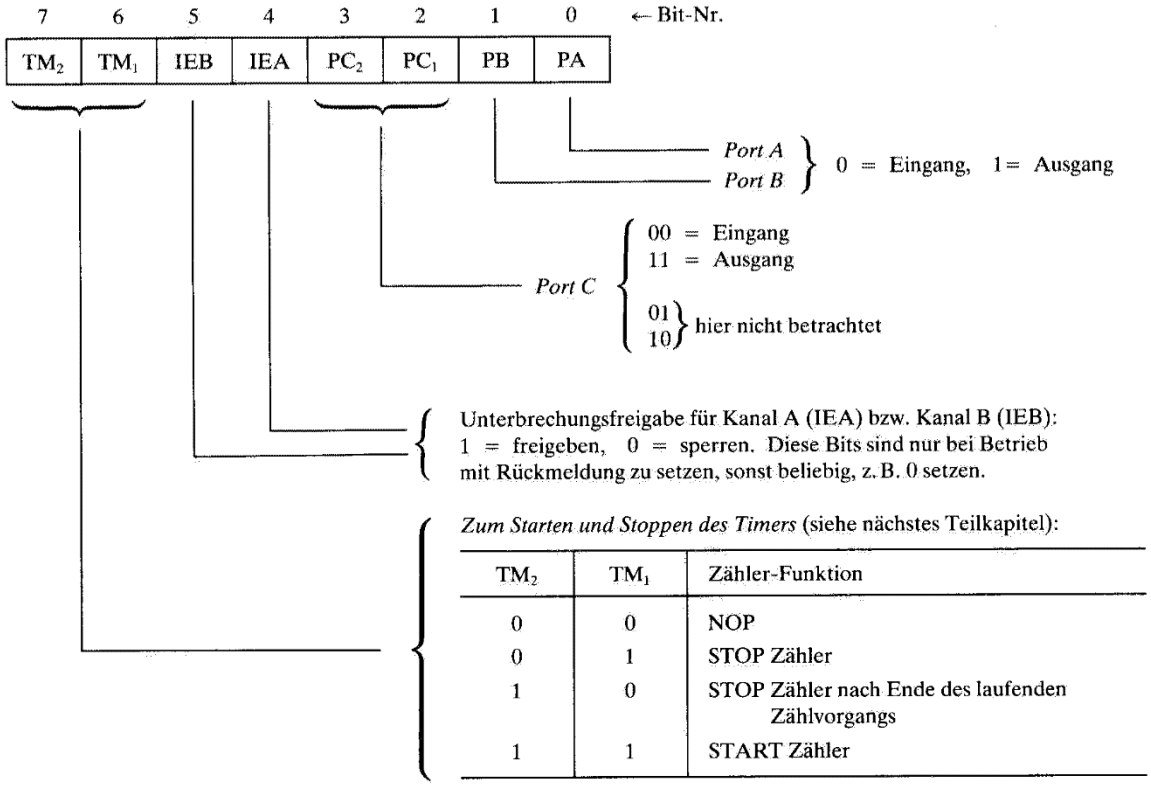
4. Ausgabe mit Rückmeldung: soll hier nicht behandelt werden.

Vor der Benutzung eines Ports in einer der vier genannten Betriebsarten mußte diese – wie gesagt – bei der Initialisierung festgelegt werden. Das dazu in das Steuerwort-Register zu übertragende Steuerwort hat die in *Bild 3.3.6* dargestellte Struktur.

Ein Beispiel: Port A und C sollen zur Informations-Ausgabe benutzt werden und Port B zum Einlesen. Wenn die vier höherwertigen, in diesem Falle bedeutungslosen, Bits gleich Null gesetzt werden, sieht das Steuerwort so aus:

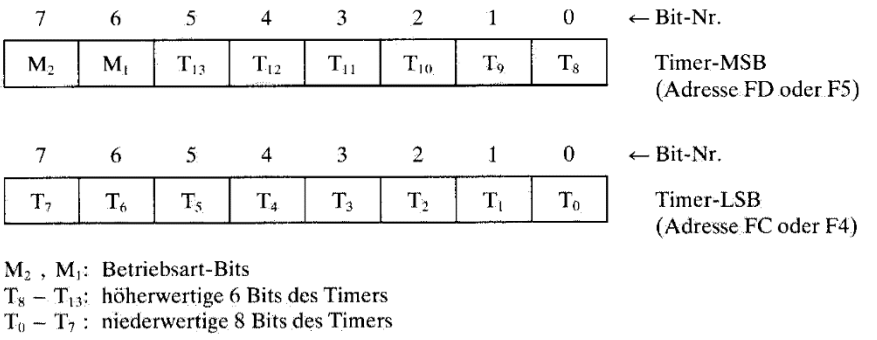
xxxx1101 bzw.: 00001101B,

d. h.: 0D H



TM_2	TM_1	Zähler-Funktion
0	0	NOP
0	1	STOP Zähler
1	0	STOP Zähler nach Ende des laufenden Zählvorgangs
1	1	START Zähler

Bild 3.3.6: Die Belegung („Programmierung") des Steuerwort-Registers für die Festlegung der Betriebsart der E/A-Ports. Der Vollständigkeit halber sind auch schon die Bits TM_2 und TM_1 aufgeführt (siehe hierzu den Text ab *Seite 30*).



M_2 , M_1: Betriebsart-Bits
$T_8 - T_{13}$: höherwertige 6 Bits des Timers
$T_0 - T_7$: niederwertige 8 Bits des Timers

Bild 3.3.7: Die Struktur der beiden Timer-Bytes.

Die gesamte Initialisierung der E/A-Ports des Bausteins 8155 Nr. 1 für diesen Betriebsfall geschieht also durch die Prozessor-Befehle (siehe *Kapitel 5*):
MVI A,0DH
(obiges Steuerwort in den Akku eingeben)
OUT F8H
(den Wert 0D aus dem Akku auf die Adresse F8 ausgeben, also ins Steuerwort-Register des 8155 Nr. 1).

Der Timer

Der Timer des 8155 ist ein 14-Bit-Abwärtszähler. Er kann mit Werten zwischen 2 und 16383 ($=2^{14}-1$) geladen werden (s. u.). Die Zählgeschwindigkeit hängt ab von der Taktfrequenz des Pulses, der auf den Timereingang (TIMER IN, Pin 3, siehe *Bild 3.3.1*) gegeben wird. Beim ECB 85 ist der Timereingang des Bausteins 8155 Nr. 1 mit dem Ausgang CLK der CPU verbunden, der einen Rechteckpuls mit 3,072 MHz liefert; die Frequenz des 6,144-MHz-Quarzes ist hier also auf die Hälfte herabgesetzt.
Die Adressen der beiden Timerbytes LSB und MSB sind bereits in *Bild 3.3.4* und *3.3.5* aufgeführt worden. Die beiden Bytes enthalten den Startwert (14 Bits, T_0-T_7 und T_8-T_{13}) und die Betriebsart; beides wird zu Beginn festgelegt (s. u.). Der Zähler-Startwert selbst belegt, wie gesagt, nur 14 Bits, so daß die höchstwertigen beiden Bits des Timer-MSB zur Festlegung der Betriebsart dienen können. Die beiden Timerbytes haben somit die in *Bild 3.3.7* dargestellte Struktur. In *Bild 3.3.8* ist die Bedeutung der beiden Betriebsart-Bits M_2 und M_1 gezeigt (diese

Betriebsart	M_2	M_1	Signal an $\overline{\text{TIMER OUT}}$ (Pin 6)
0	0	0	Während der ersten Hälfte der Zähltakte high, dann low bis zum Erreichen des Zählerstandes Null.
1	0	1	Wie bei Betriebsart 0; es wird aber beim Erreichen des Zählerstandes Null nachgeladen, so daß ein Rechteck-Signal mit dem Tastverhältnis 2 entsteht.
2	1	0	Ein low-Impuls beim Erreichen des Zählerstandes Null.
3	1	1	Ein low-Impuls beim Erreichen des Zählerstandes Null und automatisches Nachladen des Timer-Anfangswertes.

Bild 3.3.8: Die Bedeutung der Betriebsart-Bits M_2 und M_1.

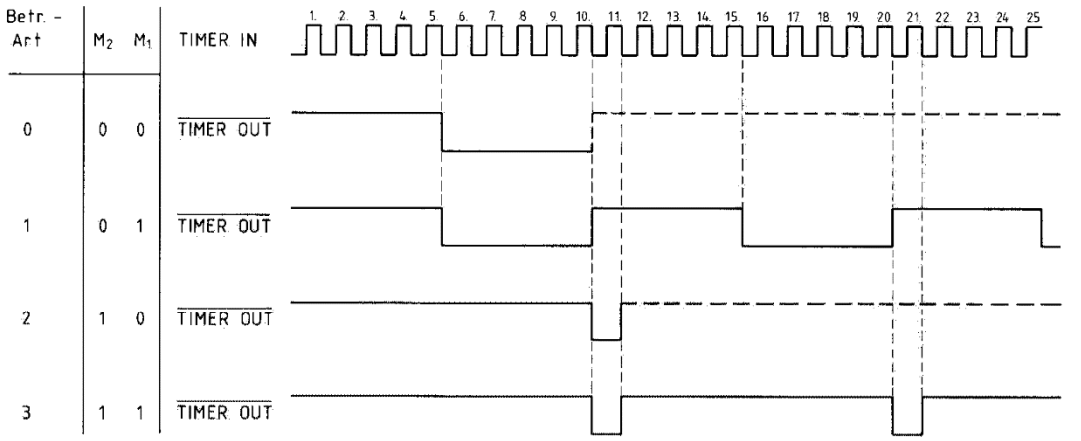


Bild 3.3.9: Das Timerausgangs-Signal für die vier Betriebsarten des 8155-Timers, wenn der Timer zuvor mit 10D (hex.: 0A) geladen wurde. Die zugehörigen Befehlsfolgen sind im Text wiedergegeben.

dürfen nicht mit den beiden Bits TM_2 und TM_1 des Steuerwortes verwechselt werden!).

Zur Programmierung des 8155-Timers Nr. 1 im ECB 85 ist insgesamt folgendes zu tun:

Zunächst müssen die beiden Timerbytes (LSB und MSB) geladen werden, und danach ist zum Starten des Timers das entsprechende Steuerwort in das Steuerwort-Register einzugeben. *Bild 3.3.9* zeigt das Ausgangssignal des Timers für die vier möglichen Betriebsarten 0, 1, 2 und 3 (vgl. *Bild 3.3.8*). Die gesamte Befehlsfolge für das Starten des Timers ist für jede Betriebsart nachstehend aufgeführt, wobei der Timer mit dem Anfangswert zehn (im Hex-Code: 0A) versehen wird.

In dieser Aufstellung ist zu beachten, daß jeweils nur das MSB des Timers zu ändern ist, weil die Betriebsart ja nur durch die höchstwertigen beiden Bits M_2 und M_1 des MSB bestimmt wird. Das Steuerwort lautet jeweils C0 H (11000000 B), weil der Timer gestartet wird ($TM_2 = 1$ und $TM_1 = 1$) und die E/A-Register des Bausteins 8155 nicht angesprochen werden sollen (vgl. hierzu noch einmal *Bild 3.3.6*).

Soll zur Bestimmung einer Zeitspanne der aktuelle Zählerstand eingelesen werden, dann ist es sinnvoll, den Timer anzuhalten und mit Hilfe eines IN-Befehls die Timerbytes zu lesen. Die Register enthalten aber keine Werte, die die Anzahl der an TIMER IN eingegangenen Impulse direkt darstellen, weil die interne Zählerschaltung in Schritten von 2 abwärts zählt; es ist also erst eine Korrektur durchzuführen.

```
Betriebsart 0:
MVI A,0A H
OUT FC H        (Laden des LSB)
MVI A,00 H
OUT FD H        (Laden des MSB)
MVI A,C0 H
OUT F8 H        (Laden des Steuerwortes)

Betriebsart 1:
MVI A,0A H
OUT FC H        (Laden des LSB)
MVI A,40 H
OUT FD H        (Laden des MSB)
MVI A,C0 H
OUT F8 H        (Laden des Steuerwortes)

Betriebsart 2:
MVI A,0A H
OUT FC H        (Laden des LSB)
MVI A,80 H
OUT FD H        (Laden des LSB)
MVI A,C0 H
OUT F8 H        (Laden des Steuerwortes)

Betriebsart 3:
MVI A,0A H
OUT FC H        (Laden des LSB)
MVI A,C0 H
OUT FD H        (Laden des MSB)
MVI A,C0 H
OUT F8 H        (Laden des Steuerwortes)
```

3.4 Die Anwender-Anschlüsse (Schnittstelle) des ECB 85

Für Experimente mit extern anzuschließenden Hardware-Schaltungen besitzt der ECB 85 eine 96polige Anwender-Schnittstelle. Hier stehen alle Prozessor-Signale (vgl. *Bild 3.1.1 und 3.1.2*) zur Verfügung und ebenso die Ein-/Ausgabe-Leitungen der beiden 8155-Bausteine (vgl. *Bild 3.3.1 und 3.3.2*). Die Belegung der 3reihigen Federleiste am ECB 85 ist in *Bild 3.4.1* aufgenommen.
Im Unterschied zu den Anschlüssen am Prozessor-Baustein 8085 A werden an der Anwender-Schnittstelle zusätzlich zum gemultiplexten Adreß/Daten-Bus D_0-D_7 die separierten Adressen A_0-A_7 sowie A_8-A_{15} zur Verfügung gestellt.
Die E/A-Ports der beiden 8155-Bausteine sind mit P1.. bzw. P2.. benannt; z. B. bedeutet P2B3 den Anschluß von *Bit 3* des *Ports B* vom 8155 Nr. 2. Des weiteren bedeuten:
TIMO-1 = TIMER OUT vom 8155 Nr. 1,
TIMO-2 = TIMER OUT vom 8155 Nr. 2,
TIMI-2 = TIMER IN vom 8155 Nr. 2.
Die Signale ANWSID und ANWSOD rühren von den auch für den Magnetband-Umsetzer benutzten Prozessor-Signalen SID und SOD her.

Stift \ Reihe	a	b	c
1	0V	0V	0V
2	+5V	+5V	+5V
3	$\overline{\text{RST5.5}}$	P1A0	A1
4	D7	P1A1	D6
5	D5	P1A2	P1A5
6	A2	P1A3	P1A4
7	A3	D4	P1A6
8	A4	D3	P1A7
9	A5	D2	P1B0
10	P1B1	D1	D0
11	A6	P1B2	ALE
12	A7	P1B3	$\overline{\text{WR}}$
13	A0	P1B4	$\overline{\text{RST6.5}}$
14	$\overline{\text{RD}}$	P1B5	$\overline{\text{TIMO-1}}$
15	IO/$\overline{\text{M}}$	P1B6	P1C5
16	A8	P1B7	$\overline{\text{TRAP}}$
17	RESOUT	P1C0	INTR
18	CLK	P1C1	HOLD
19	P1C4	P1C2	P1C3
20	$\overline{\text{INTA}}$	S1	A9
21	P2A0	ANWRDY	A10
22	P2A1	$\overline{\text{TIMO-2}}$	A11
23	P2A2	A13	A12
24	P2A3	A15	A14
25	P2A4	S0	P2A5
26	P2A6	$\overline{\text{RESIN}}$	P2A7
27	P2B1	P2B0	P2B2
28	P2B3	P2C5	P2B4
29	P2B6	P2C4	P2B7
30	HLDA	P2B5	P2C0
31	P2C1	TIMI-2	ANWSID
32	P2C3	ANWSOD	P2C2

Bild 3.4.1: Die Signale der Anwender-Schnittstelle des ECB 85. Nähere Erläuterungen im Text.

4. Zur Programmierung des 8085 A

4.1 Die Adressierungsarten

Beim Ablauf von Programmen werden im Mikrocomputer stets – wie schon früher gesagt – Daten miteinander verknüpft bzw. nach entsprechenden Anweisungen verarbeitet. Die Datenbytes können dabei entweder in einem Register der CPU *(Bild 3.1.3)* oder auch im Speicher *(Bild 3.2.1)* stehen bzw. von oder nach dort übertragen werden; und schließlich können Daten auch über die E/A-Kanäle mit der Peripherie ausgetauscht werden (siehe *Bild 3.3.5*).
Es muß also bei praktisch jedem Befehl (außer z. B. bei Steuerbefehlen wie NOP oder HLT) eine Aussage darüber gemacht werden, woher und/oder wohin Daten transportiert werden sollen bzw. auf welches Register oder welchen Speicherplatz der Befehl wirken soll. Mit anderen Worten: nahezu jeder Befehl ist mit einer *Adressierung* verbunden. Die verschiedenen Möglichkeiten der Adressierung beim 8085 sollen hier kurz vorgestellt werden.

Direkte Speicher-Adressierung

Das Befehlswort enthält explizit („ausdrücklich") die Adresse des anzusprechenden Speicherplatzes.

Beispiel:

LDA 1832H	d. h. lade den Inhalt der Speicherzelle mit der (im Hex.-Code anzugebenden) Adresse 1832 in den Akkumulator.
3A 32 18	ist der Hex.-Code für diesen Befehl.

An dieser Stelle sei auf einen wichtigen Punkt aufmerksam gemacht: Bei der Umsetzung vom Mnemo-Code (LDA 1832H) in den hexadezimalen Maschinen-Code müssen die beiden Adreßbytes vertauscht werden; im Hex.-Code steht also das niederwertige Adreßbyte immer vorn!
Ebenfalls mit der direkten Adressierung wird bei den IN- und OUT-Befehlen gearbeitet, bei denen die E/A-Adresse Bestandteil des Befehls ist; z. B.: OUT F9H, hexadezimal: D3 F9, d. h. Ausgabe des im Akkumulator befindlichen Datenbytes auf die E/A-Adresse F9 (Port A des 8155 Nr. 1, siehe *Bild 3.3.5*).

Indirekte Adressierung

Sie wird auch *Registerpaar-Adressierung* oder *indirekte Register-Adressierung* genannt. Die Speicheradresse ist hier durch den Inhalt eines Registerpaares, meist des Registerpaares H/L, festgelegt; in diesem Falle enthält das Register H („high") das höherwertige Byte und das Register L („low") das niederwertige Byte der Speicherplatz-Adresse. Der Speicherplatz wird im Mnemo-Code dabei mit M abgekürzt.

Beispiel:

MOV A,M	d. h. lade den Inhalt des durch H,L adressierten Speicherplatzes in den Akkumulator (A). Wenn zuvor H mit 18 und L mit 32 geladen wurde, wird also die Speicherplatz-Adresse 1832 angesprochen.
7E	ist die hexadezimale Codierung des Befehls MOV A,M; er umfaßt also nur 1 Byte.

Direkte Register-Adressierung

Der Befehl gibt direkt die Register an, die am Datentransport beteiligt sind.

Beispiel:

MOV A,B	d. h. bringe den Inhalt des Registers B in den Akkumulator (A).
78	ist die hexadezimale Codierung dieses Befehls.

Unmittelbare Adressierung

Der Befehl enthält einen Zahlenwert (eine Konstante, 1 Byte), der unmittelbar in das angesprochene Register geladen wird.

Beispiel:

MVI A,2AH	d. h. lade unmittelbar (move immediately) den Wert 2AH = 42D in das Register A.
3E 2A	ist die hexadezimale Codierung.

Adressierung durch den Stackpointer (Kellerspeicher-Zeiger)

Jeder Speicherplatz kann auch über das 16-Bit-Stackpointer-Register adressiert werden. Außer bei Unterprogramm-Aufrufen (s. u.) trifft dies bei den Befehlen PUSH und POP zu. Mit diesen Befehlen kann der Inhalt von Registerpaaren der CPU (z. B. B/C, D/E, H/L oder auch PSW, siehe *Kapitel 4.2*) abgelegt und später wieder zurückgeholt werden.
Die PUSH-Operation bringt 16 Datenbits (2 Bytes) aus einem Registerpaar oder dem Befehlszähler in den Kellerspeicher, d. h. in eine durch den Stackpointer bestimmte RAM-Adresse, und zwar:
die höherwertigen 8 Bits in die Adresse SP – 1,
die niederwertigen 8 Bits in die Adresse SP – 2.
Ein Beispiel: Es sei der Stackpointer SP mit 1D00H geladen, das Register B mit 6AH und das Register C mit 30H. Dann bewirkt PUSH B, daß 6AH in die Speicherzelle 1BFF und 30H in die Speicherzelle 1BFE abgespeichert wird.
Die POP-Operation bringt 2 Byte aus dem Kellerspeicher in ein Registerpaar. Dabei wird der Inhalt der Adresse SP als niederwertiges Byte behandelt und der Inhalt von SP + 1 als höherwertiges Byte.
Wenn SP mit 1BFE geladen war sowie die Speicherzelle 1BFE mit 00H und 1BFF mit 18H, dann bewirkt POP H, daß das Register L mit 00H und das Register H mit 18H geladen werden.
Bei der Benutzung der Befehle PUSH und POP ist auf die Reihenfolge zu achten, in der sie benutzt werden: Die Daten, die zuletzt mit PUSH abgelegt wurden, werden auch zuerst wieder mit POP aus dem Kellerspeicher herausgeholt. Daraus ergibt sich auch, daß in der Regel beide Befehle gleich oft vorkommen.

Unterprogramme

Beim Aufruf von Unterprogrammen wird ebenfalls der Kellerspeicher benutzt. Unterprogramme werden mit dem CALL-Befehl aufgerufen; z. B. lautet er CALL 1900H. Dies bewirkt einen Sprung zu einem Programmteil (Unterprogramm), der bei der Adresse 1900 beginnt. Bei der Rückkehr aus dem Unterprogramm (RETURN-Befehl) muß auf den Befehl im Hauptprogramm (in dem der CALL-Befehl steht) gesprungen werden, der auf den CALL-Befehl folgt. Die Adresse dieses Befehls wurde bei der Ausführung des CALL-Befehls im Kellerspeicher abgelegt.

4.2 Das Prozessor-Statuswort PSW

Das Prozessor-Statuswort besteht aus dem Akkumulator und dem Flagbyte. Es kann, ähnlich wie für die Registerpaare der CPU im vorangegangenen Abschnitt gezeigt, mit dem Befehl PUSH PSW in den Kellerspeicher abgelegt und mit POP PSW wieder zurückgeholt werden. Dies wird häufig ausgenutzt, um die im Flagbyte enthaltene Information auszuwerten. Der Akkumulator bildet das höherwertige Byte des PSW.
Denn für die Programmierung ist es wichtig, das Flagbyte zu kennen. Es dient mit seinen 5 Flagbits zur Kennzeichnung des Resultats einer Operation, so daß durch Abfrage dieser 5 Bits (s. u.) je nach Ergebnis der vorangegangenen Operation im Programm zu unterschiedlichen Befehlen gesprungen werden kann. *Bild 4.2.1* zeigt den Aufbau des Flagbytes und die Bedeutung der 5 Flagbits.
Es gilt für jedes Flagbit die Festlegung:
Bit = 1: Bit gesetzt,
Bit = 0: Bit nicht gesetzt.
Das *Carry-Bit* ist gesetzt, wenn z. B. bei einer Addition das Ergebnis größer als 255D ist.
Das *Hilfscarry-Bit* beeinflußt die Funktion des DAA-Befehls und wird gesetzt, wenn z. B. bei einer Addition ein Übertrag von Bit 3 nach Bit 4 erfolgt.
Das *Vorzeichen-Bit* ist gesetzt, wenn nach einer arithmetischen oder logischen Operation das Bit 7 (also das höchstwertige Bit) des Ergebnisses 1 ist.
Das *Null-Bit* ist gesetzt, wenn das Ergebnis einer Operation Null ist.
Das *Paritäts-Bit* ist gesetzt, wenn die Anzahl der gesetzten Bits des Ergebnisses gerade ist.
Dabei ist zu beachten, daß sich das Ergebnis einer jeden Operation stets im Akkumulator befindet.

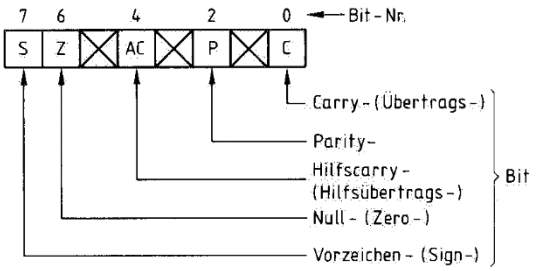


Bild 4.2.1: Der Aufbau des Flagbytes.

5. Die Befehlsworte für den Mikroprozessor 8085 A

5.1 Transferbefehle für 8-Bit-Daten

Transferbefehle bringen Daten aus einem Register in ein anderes oder Daten aus dem Speicher in ein Register oder umgekehrt. Dabei bleibt die Information des Senders stets erhalten. Nach Ausführung des Befehls hat der Empfänger also dieselbe Information wie der Sender. Es können auch Direktwerte (d. h. Konstanten; vgl. unmittelbare Adressierung) übertragen werden.
Jedem der 8 Register der CPU (vgl. *Bild 3.1.3*) ist beim Aufbau der Transferbefehle eine 3-Bit-Adresse fest zugeordnet; siehe Tabelle. Dabei werden die 3 Bits allgemein mit ddd (d für engl. destination = Bestimmung, Ziel) bzw. sss (s für engl. source = Quelle) abgekürzt:

Register	ddd/sss
B	000
C	001
D	010
E	011
H	100
L	101
M	110
A	111

MVI r_1, r_2	die Information des Registers r_2 wird in das Register r_1 übertragen, ohne daß der Inhalt von r_2 verändert wird. r_1 und r_2 können irgendwelche Register der CPU sein.
01dddsss	ist die binäre Darstellung des Befehls MOV r_1, r_2. Darin ist ddd die Adresse des Empfänger-(Ziel-)Registers r_1, und sss ist die Adresse des Sende-(Quell-)Registers r_2, (siehe Tabelle).

So ergibt sich dann für den Befehl
MOV C,A
01001111 als Binärmuster,
4F als Hex.-Code

Damit der Programmierer nicht jedes einzelne Bitmuster nach diesem Beispiel selbst entwickeln muß, ist in der Befehlsliste im Anhang für alle MOV-Befehle der Hex.-Code direkt angegeben.

MVI r,D8	bewirkt, daß ein 8-Bit-Datenwort (hier mit D8 symbolisiert) in das Register r gebracht wird.
00ddd110	ist die binäre Darstellung des Befehls MVI r. Für ddd ist die 3-Bit-Zieladresse des anzusprechenden Registers gemäß der Tabelle einzusetzen.

Soll z. B. das Datenbyte 5A in den Akkumulator geladen werden, dann ist
ddd = 111,
und der vollständige Befehl lautet:
MVI A,5A
00111110 01011010 binär,
3E 5A Hex.-Code
Das wichtigste Register des 8085 ist der Akkumulator (vgl. *Kapitel 3.1*). Er ist Sende- bzw. Empfangs-Register bei Aus- bzw. Eingabe (z. B. mit den E/A-Ports des 8155), und er enthält bei arithmetischen oder logischen Operationen den einen Operanden und am Ende der Operation deren Ergebnis. Daher gibt es einige spezielle Transferbefehle nur für den Akkumulator.

LDA adr	lädt den Inhalt der Speicherzelle mit der 16-Bit-Adresse adr in den Akku.
3A XXXX	Hex.-Code (XXXX sind die beiden Adreßbytes).

STA adr	speichert den Inhalt des Akkus in die Speicherzelle mit der 16-Bit-Adresse adr.
32 XXXX	Hex.-Code (XXXX = Adresse).

Alle Register, auch der Akkumulator, können mit der Speicherzelle M arbeiten, deren Adresse im Registerpaar HL steht (indirekte Adressierung, vgl. *Kapitel 4.1*). Für den Akkumulator gibt es solche Transferbefehle auch zu oder von solchen Speicherplätzen, deren Adresse im Registerpaar BC oder DE steht.

Für das Einholen eines Datenbytes von einem Speicherplatz stehen hier folgende zwei Befehle zur Verfügung:

LDAX B	der Inhalt des Speicherplatzes, dessen Adresse in BC steht, wird in den Akku gebracht.
0A	Hex.-Code
LDAX D	der Inhalt des Speicherplatzes, dessen Adresse in DE steht, wird in den Akku gebracht.
1A	Hex.-Code

Zum Abspeichern eines Datenbytes vom Akku in den Speicher dienen die STAX-Befehle:

STAX B	der Akku-Inhalt wird unter der Adresse abgespeichert, die im Registerpaar BC steht.
02	Hex.-Code
STAX D	der Akku-Inhalt wird unter der Adresse abgespeichert, die im Registerpaar DE steht.
12	Hex.-Code

Für die Ein- und Ausgabe über periphere Bausteine (z. B. 8155, siehe Kapitel 3.3) gibt es die Befehle IN und OUT.

IN EA-adr	die Information, die unter der 8-Bit-E/A-Adresse EA-adr anliegt, wird in den Akkumulator gebracht.
DB XX	Hex.-Code (XX = EA-Adr.) *Beispiel:* IN FB (hex.: DB FB) holt die an Port C des 8155 Nr. 1 anstehende Information in den Akkumulator.
OUT EA-adr	der Inhalt des Akkus wird unter der E/A-Adresse EA-adr ausgegeben.
D3 XX	Hex.-Code *Beispiel:* OUT F9 (hex.: D3 F9) gibt das im Akku befindliche Byte an Port A des 8155 Nr. 1 aus.

5.2 Transferbefehle für 16-Bit-Daten

Entsprechend dem 1-Byte-Transfer, der mit Hilfe der MVI-Befehle durchgeführt wird, gibt es die LXI-Befehle für die Übertragung von 16-Bit-Daten. Es gibt vier 16-Bit-Transferbefehle:

LXI B,D16	das Datenwort D16 wird ins Registerpaar BC gebracht.
01 XXXX	Hex.-Code (XXXX = 16-Bit-Datenwort).
LXI D,D16	das Datenwort D16 wird ins Registerpaar DE gebracht.
11 XXXX	Hex.-Code (XXXX = 16-Bit-Datenwort).
LXI H,D16	das Datenwort D16 wird ins Registerpaar HL gebracht.
21 XXXX	Hex.-Code (XXXX = 16-Bit-Datenwort).
LXI SP,D16	das Datenwort D16 wird in den Stackpointer gebracht.
31 XXXX	Hex.-Code (XXXX = 16-Bit-Datenwort).

Bei der Bearbeitung von 16-Bit-Wörtern spielt das Registerpaar HL eine besondere Rolle. Wie für den Akkumulator bei 8-Bit-Verarbeitung gibt es hier für das Registerpaar HL besondere Befehle zum direkten Laden bzw. Abspeichern:

LHLD adr	das Registerpaar HL wird direkt mit dem Inhalt der Speicherzellen adr und adr+1 geladen, und zwar gelangt der Inhalt von adr+1 nach H (H für höherwertig), und der Inhalt von adr nach L (L für lower = niederwert.)
2A XXXX	Hex.-Code (XXXX = Adresse).
SHLD adr	der Inhalt von HL wird ausgespeichert: Inhalt von H nach adr+1, und Inhalt von L nach adr.
22 XXXX	Hex.-Code (XXXX = Adresse).
XCHG	der Inhalt von HL wird mit dem von DE vertauscht, und zwar H mit D und E mit L.
EB	Hex.-Code für den Befehl XCHG.

Auch die PUSH- und POP-Befehle (vgl. *Kapitel 4.1*) übertragen 16-Bit-Daten:

PUSH rp	der Inhalt des Registerpaares rp wird ausgespeichert;
POP rp	das Registerpaar rp wird mit dem Inhalt von Stackpointerzellen geladen.

Am Beispiel des Registerpaares BC (abgekürzt üblicherweise als B) wird der genaue Ablauf der beiden Befehle nachstehend erläutert.

PUSH B	der Inhalt des Registers B gelangt in die Speicherzelle mit der Adresse SP−1, und der Inhalt von C gelangt zur Speicherzelle SP−2. Dabei ist SP die Adresse, die gerade im Stapelzeiger (Stackpointer, Kellerspeicherzeiger) steht.
POP B	der Inhalt der Speicherzelle, deren Adresse gerade im Stackpointer (SP) steht, wird in das Register C geladen, und der Inhalt der Speicherzelle SP+1 in das Register B.

Bei der Ausführung der Befehle PUSH und POP ist auf die Reihenfolge zu achten; vgl. *Kapitel 4.1.* Nachfolgend sind kurz die möglichen PUSH- und POP-Befehle aufgeführt, wobei der Hex.-Code in Klammern steht:

PUSH B	(C5)	POP B	(C1)
PUSH D	(D5)	POP D	(D1)
PUSH H	(E5)	POP H	(E1)
PUSH PSW	(F5)	POP PSW	(F1)

5.3 Arithmetische Operationen für 8-Bit-Daten

Bei arithmetischen Operationen werden die Operanden stets als binäre Zahlen aufgefaßt. Die Befehlsworte für den Prozessor 8085 ermöglichen das Addieren oder Subtrahieren von Binärzahlen; alle anderen Rechenarten müssen durch Programme realisiert werden.
Bei Additionen oder Subtraktionen steht immer ein Operand im Akkumulator; der andere Operand kann angesprochen werden durch:
unmittelbare Adressierung (z. B. ADI 03),
direkte Register-Adressierung (ADD B), oder
indirekt über das Reg.-Paar HL (ADD M).
Nach der Berechnung steht das Ergebnis im Akkumulator. Es beeinflussen nicht alle arithmetischen Befehle alle Flag-Bits (vgl. hierzu *Kapitel 4.2*); aus der Tabelle aller 8085-Befehle im Anhang ist ersichtlich, welche Flag-Bits wann geändert werden.
Die Befehle, die die Register-Inhalte inkrementieren (um 1 erhöhen) oder dekrementieren (um 1 erniedrigen), benutzen den Akkumulator nicht und setzen bzw. rücksetzen auch das Carry-Bit nicht.
Bei den Additions- und Subtraktions-Befehlen unterscheidet man zwei Arten: zum einen solche, die das Carry-Bit vorhergegangener Operationen verarbeiten, und zum anderen solche, die dies nicht tun. Man nutzt diese Tatsache bei der Bearbeitung von Datenwörtern, die länger als 8 Bit sind.
Nachstehend werden die 8-Bit-Additions- und Subtraktions-Befehle des 8085 aufgeführt. Zur Adressierung der Register mit ddd bzw. sss siehe auch hier die Tabelle auf *Seite 34.*

ADD r	addiert zum Inhalt des Akkus (A) den Inhalt des Registers r; Ergebnis in A: A + r → A
10000sss	binär
	Beispiel:
	ADD M (für M ist sss = 110)
	10000110 binär
	86 Hex.-Code
ADC r	addiert zum Akku-Inhalt den Inhalt von r und das Carry-Bit; Ergebnis in A:
	A + r + carry → A
10001sss	binär
	Beispiel:
	ADC M (sss = 110)
	10001110 binär
	8E Hex.-Code
ADI D8	addiert zum Akku-Inhalt das 8-Bit-Datenwort D8; Ergebnis in A: A + D8 → A
C6 XX	Hex.-Code (XX = Datenwort)
	Beispiel:
	ADI 02 (A + 02 → A)
	C6 02 Hex.-Code

ACI D8	addiert zum Akku-Inhalt das Datenwort D8 und das Carry-Bit; Ergebnis in A: A + D8 + carry → A
CE XX	Hex.-Code (XX = Datenwort) *Beispiel:* ACI 02 (A + 2 + carry → A) CE 02 Hex.-Code

SUB r	subtrahiert den Inhalt des Registers r vom Akku-Inhalt Ergebnis im Akku: A – r → A
10010sss	binär *Beispiel:* SUB M (sss = 110) 10010110 binär 96 Hex.-Code

SBB r	subtrahiert außer dem Inhalt von r auch das Carry-Bit vom Akku-Inhalt; Ergebnis im Akku: A – M – carry → A
10011sss	binär *Beispiel:* SBB M (sss = 110) 10011110 binär 9E Hex.-Code

SUI D8	subtrahiert direkt das 8-Bit-Datenwort D8 vom Akku; Ergebnis im Akku: A – D8 → A
D6 XX	Hex.-Code (XX = Datenwort) *Beispiel:* SUI 08 (A – 8 → A) D6 08 Hex.-Code

SBI D8	subtrahiert direkt das 8-Bit-Datenwort D8 und das Carry-Bit vom Akku-Inhalt; Ergebnis im Akku: A – D8 – carry → A
DE XX	Hex.-Code (XX = Datenwort) *Beispiel:* SBI 08 (A – 8 – carry → A) DE 08 Hex.-Code

Es ist zu beachten, daß der Prozessor 8085 bei der Subtraktion das Carry-Bit setzt, wenn *kein* Übertrag entsteht, wenn er sich also etwas „leihen" mußte. Bei der Addition wird das Carry-Bit aber gesetzt, wenn ein Übertrag entsteht.

Für die Register des 8085 gibt es auch eigene Befehle zum Inkrementieren und zum Dekrementieren (Addition bzw. Subtraktion von 1):

INR r	Inhalt des Registers r wird um 1 erhöht; der Akku-Inhalt wird *nicht* verändert: r + 1 → r
00ddd100	binär *Beispiel:* INR M (sss = 110) 00110100 binär 34 Hex.-Code

DCR r	der Inhalt des Registers r wird um 1 erniedrigt; Akku-Inhalt bleibt: r – 1 → r
00ddd101	binär *Beispiel:* DCR M (sss = 110) 00110101 binär 35 Hex.-Code

5.4 Arithmetische Operationen für 16-Bit-Daten

Bei der Verarbeitung von 16-Bit-Daten müssen jeweils Registerpaare angesprochen werden (vgl. dazu *Kapitel 3.2*).

Befehle zum Inkrementieren und Dekrementieren:

INX rp	inkrementiert den Inhalt des Registerpaares rp; der Akku-Inhalt bleibt unverändert: rp + 1 → rp

DCX rp	dekrementiert den Inhalt des Registerpaares rp; der Akku-Inhalt bleibt unverändert: rp – 1 → rp

Die möglichen Befehle zum Inkrementieren und zum Dekrementieren von Registerpaaren lauten (Hex.-Code jeweils in Klammern):

INX B	(03)	DCX B	(0B)
INX D	(13)	DCX D	(1B)
INX H	(23)	DCX H	(2B)
INX SP	(33)	DCX SP	(3B)

Diese Befehle verändern kein Flagbit!

Befehle zur Addition:

DAD rp	der Inhalt des Registerpaares rp wird zum Inhalt von HL addiert; das Ergebnis wird in HL gespeichert. rp + HL → HL

Es sind folgende 16-Bit-Additionsbefehle möglich (Hex.-Code in Klammern):

DAD B	(09)	DAD D	(19)
DAD H	(29)	DAD SP	(39)

Ein Spezialfall tritt hier auf, da das Ergebnis im Registerpaar HL gespeichert wird: der Befehl DAD H verdoppelt den Inhalt von H, d. h. verschiebt ihn um eine Stelle nach links (es werden ja Binärzahlen verarbeitet), wobei eine Null nachgezogen wird.

5.5 Logische Operationen: UND, ODER, EXOR, Negation

Die zulässigen Adressierungs-Arten bei logischen Verknüpfungen sind dieselben wie bei den arithmetischen Operationen.
Alle hier aufgeführten logischen Operationen setzen das Carry-Bit zurück!

UND-Verknüpfung:

ANA r	der Akku-Inhalt wird mit dem Inhalt des Registers r Bit für Bit UND-verknüpft. Das Ergebnis steht im Akku.
10100sss	binär
	Beispiel: ANA A 10100111 binär (sss = 111) A7 (Hex.-Code) der Akku-Inhalt wird mit sich selbst UND-verknüpft, also nicht verändert; aber es wird dabei das Carry-Bit gelöscht.
ANI D8	der Akku-Inhalt und das Datenwort D8 werden Bit für Bit UND-verknüpft; das Ergebnis steht im Akku.
E6 XX	Hex.-Code (XX = Datenwort) *Beispiel:* ANI 0F E6 0F (Hex.-Code)

Der ANI-Befehl wird z. B. zum Ausblenden („Maskieren") von Bits benutzt. Durch ANI 0F bleiben beispielsweise die niederwertigen 4 Bits erhalten wegen der UND-Verknüpfung mit dem niederwertigen Halbbyte F H = 1111 B, und die höherwertigen 4 Bits werden „gelöscht" wegen der UND-Verknüpfung mit dem höherwertigen Halbbyte 0 H = 0000 B.

ODER-Verknüpfung:

ORA r	der Akku-Inhalt und der Inhalt des Registers r werden Bit für Bit ODER-verknüpft. Das Ergebnis steht im Akku.
10110sss	binär *Beispiel:* ORA A 10110111 binär (sss = 111) B7 Hex.-Code Dieser Befehl ORA A verändert am Akku-Inhalt nichts, löscht aber das Carry-Bit.
ORI D8	der Akku-Inhalt und das Datenwort D8 werden Bit für Bit ODER-verknüpft, d. h. alle gesetzten Bits aus beiden Bytes bleiben im Ergebnis-Byte gesetzt; dieses befindet sich dann im Akku.
F6 XX	Hex.-Code (XX = Datenwort)

EXOR-Verknüpfung:

XRA r	der Akku-Inhalt und der Inhalt des Registers r werden Bit für Bit Exklusiv-ODER-verknüpft, d. h. im Ergebnis-Byte (das dann im Akku steht) ist jeweils an der Stelle das Bit gesetzt, an der in *einem* der beiden Bytes das Bit gesetzt war.
10101sss	binär *Beispiel:* XRA A 10101111 binär (sss = 111) AF Hex.-Code Dieser Befehl XRA A löscht den Akkumulator, der danach also 00 H enthält.
XRI D8	der Akku-Inhalt und das Datenwort D8 werden EXOR-verknüpft (Definition siehe beim Befehl XRA r); das Ergebnis steht im Akku.
EE XX	Hex.-Code (XX = Datenwort)

Negation:

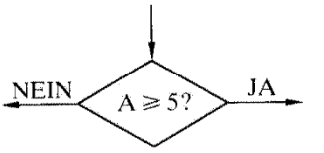
CMA	der Akku-Inhalt wird Bit für Bit invertiert (negiert); das Ergebnis steht danach im Akku.
2F	Hex.-Code

5.6 Vergleichs-Operationen

Auch Vergleichs-Operationen sind beim 8085 möglich. Dabei wird das durch den Befehl angesprochene Byte als Zahl interpretiert und mit dem Akku-Inhalt verglichen, und zwar wird das betr. Byte vom Akku-Inhalt subtrahiert; hierbei wird das Resultat aber *nicht* im Akku gespeichert. Über das Subtraktions-Ergebnis informiert das Flagbyte (vgl. *Kap. 4.2*), das über bedingte Sprünge (s. u.) ausgewertet werden kann.

CMP r	der Inhalt des Registers r wird mit dem Akku-Inhalt verglichen.
10111sss	binär
CPI D8	das 8-Bit-Datenwort D8 wird mit dem Akku-Inhalt verglichen.
FE XX	Hex.-Code (XX = Datenwort)

Beispiel:
Überprüfung des Inhalts von Register C darauf, ob er größer oder gleich 5 ist:




```
MOV A,C
CPI 05
JNC JA
NEIN: ...
JA:   ...
```

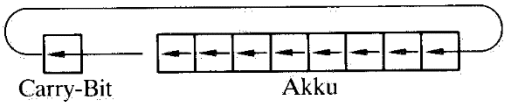
Steht in C (bzw. nach dem MOV-Befehl im Akku) eine Zahl größer/gleich 5, so ergibt die Subtraktion beim CPI-Befehl eine Zahl größer/gleich 0: das Carry-Bit wird rückgesetzt. Der bedingte Sprung JNC wird dann ausgeführt und das Label JA erreicht. Andernfalls ist die Zahl im Akku kleiner als 5, und es entsteht bei der Addition des Komplements kein Übertrag: das Carry-Bit wird gesetzt.

5.7 Rotations-Befehle

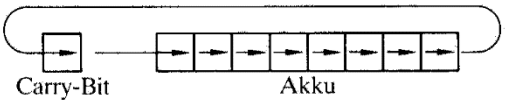
Bei jedem der nachfolgend beschriebenen Rotationsbefehle RAL, RAR, RLC und RRC erfolgt jeweils eine Verschiebung des Akku-Inhalts um 1 Bit nach rechts bzw. nach links. Unterschiedlich ist dabei die Behandlung des Carry-Bits. Die Skizzen zeigen deutlich, was bei der Ausführung des Befehls geschieht; in Klammern ist der Hex.-Code angegeben.

Durch anschließende Abfrage des Carry-Bits eignen sich diese Befehle zur Überprüfung, ob ein bestimmtes Bit des Akkus gesetzt war oder nicht.

RAL (17)	(rotate accumulator left) Akku-Inhalt rotiert links herum durch das Carry-Bit.

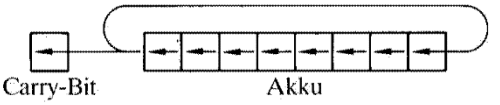


RAR (1F) (rotate accumulator right)
Akku-Inhalt rotiert rechts herum durch das Carry-Bit.

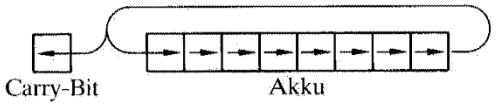


Bei den Befehlen RLC und RRC wird nicht durch das Carry-Bit rotiert, sondern zusätzlich in das Carry-Bit hineingeschrieben:

RLC (07) Akku-Inhalt rotiert links herum.



RRC (0F) Akku-Inhalt rotiert rechts herum



5.8 Sprungbefehle, Unterprogramme und Unterbrechungen

5.8.1 Sprungbefehle

Sprungbefehle ermöglichen den Aufbau von Programmschleifen; bei bedingten Sprüngen werden vor der Ausführung bestimmte Flag-Bits abgefragt. Verfolgt man den weiteren Ablauf des Programms (Ziel des Sprungs), so kennt man den Zustand des abgefragten Flag-Bits (Zustands-Bits). Der jeweilige Wert des Zustands-Bits wird also beim bedingten Sprung als Entscheidungs-Merkmal benutzt.
In allen Sprungbefehlen (außer bei PCHL, s. u.) ist die 16-Bit-Adresse adr enthalten; sie gibt an, wo das nächste Befehlswort steht, das im Falle des Sprunges auszuführen ist.
Wenn ein Sprungbefehl in das Befehlsregister der CPU und in die Hilfsregister W und Z aufgenommen wurde und bei bedingten Sprüngen die Bedingung erfüllt ist, dann gibt der Prozessor nicht den Inhalt des Befehlszählers auf den Adreß-Bus, sondern die soeben erhaltene Adresse adr aus den Hilfsspeichern W und Z.
Nach Ausführung des Sprungbefehls enthält der Befehlszähler (PC) die auf adr folgende Adresse. Damit „weiß" der Prozessor nun auch nicht mehr, von wo aus der Sprung ausgeführt wurde, und das Programm wird einfach bei adr+1 fortgesetzt. Anders ist dies beim Aufruf von Unterprogrammen (s. u.).
Bei den nachfolgend aufgeführten Befehlen bedeutet XXXX jeweils die hexadezimal anzugebende Adresse, zu der der Sprung führen soll.

Unbedingte Sprünge:

JMP adr	springe (jump) zu der Adresse adr, ohne die Erfüllung irgendeiner Bedingung abzufragen
C3 XXXX	Hex.-Code
PCHL	das Programm springt zu der Adresse, die im Registerpaar HL steht; PCHL bewirkt nämlich, daß der Inhalt von HL in den Programmzähler PC geladen wird.
E9	Hex.-Code

In der nachfolgenden Tabelle der bedingten Sprünge sind folgende Abkürzungen verwendet (vgl. zum Flag-Byte noch einmal Kapitel 4.2):
C für carry (Übertrag)
Z für zero (Null)
E für even (gerade)
O für odd (ungerade)
P für positiv
M für minus

Bedingte Sprünge:

	springe zur Adresse adr, (hex.: XXXX), wenn ...
JC adr	Carry-Bit gesetzt
DA XXXX	
JNC adr	Carry-Bit zurückgesetzt
D2 XXXX	
JZ adr	Null-Bit gesetzt
CA XXXX	
JNZ adr	Null-Bit zurückgesetzt
C2 XXXX	
JM adr	Vorzeichen-Bit gesetzt
FA XXXX	(Ergebnis negativ)
JP adr	Vorzeichen-Bit zurückgesetzt
F2 XXXX	(Ergebnis positiv)
JPE adr	Paritäts-Bit gesetzt
EA XXXX	
JPO adr	Paritäts-Bit zurückgesetzt
E2 XXXX	

5.8.2 Aufruf von Unterprogrammen (UP)

Soll eine Befehlsfolge innerhalb eines Programms mehrmals durchlaufen werden, dann läßt man sie in Form eines Unterprogramms (abgekürzt: UP) ablaufen, um Speicherplatz zu sparen (weil die Befehlsfolge dann nur einmal im Speicher steht).
Der Aufruf eines Unterprogramms ist ein bedingter oder ein unbedingter Sprung zur Anfangs-Adresse des UP. Jedes Unterprogramm endet mit der Rückkehr ins Hauptprogramm (Befehl RET, return, s. u.). Um diese Rückkehr zu ermöglichen, müssen beim Sprung (beim Aufruf) zusätzliche Maßnahmen zur Rettung der Rücksprung-Adresse getroffen werden. Mit anderen Worten: Es muß „irgendwo" festgehalten werden, von wo aus der Sprung erfolgte, damit der weitere Ablauf des Hauptprogramms möglich ist, wenn das Unterprogramm abgearbeitet wurde. Genaugenommen wird die Adresse gespeichert, die im Hauptprogramm auf den UP-Aufruf (Absprung ins UP) folgt.

Wenn in einem Programm Unterprogramme aufzurufen sind, dann muß vorher (bei der sogen. Programm-Initialisierung) der Kellerspeicher-Zeiger (Stackpointer) mit einer RAM-Adresse geladen werden; dabei ist diese Adresse so zu wählen, daß von ihr aus abwärts noch genügend Speicherstellen frei sind zur Aufbewahrung von Rückkehr-Adressen.

Aufruf und Rücksprung (unbedingt):

CALL adr	rufe das bei der Adresse adr beginnende UP auf
CD XXXX	Hex.-Code
RET	kehre ins Hauptprogramm zurück
C9	Hex.-Code

Ebenso wie bei den Sprung-Befehlen (siehe vorigen Abschnitt 5.7.1) gibt es auch bei Aufruf von UP und bei der Rückkehr ins Hauptprogramm die Abfrage auf bestimmte Flag-(Zustands-)Bits. Die im vorigen Abschnitt genannten Abkürzungen C, Z, E, O, P und M gelten hier entsprechend.

Bedingter Aufruf von UP

	rufe das bei Adresse adr (hex.: XXXX) beginnende UP auf, wenn ...
CC adr	Carry-Bit gesetzt
DC XXXX	
CNC adr	Carry-Bit zurückgesetzt
D4 XXXX	
CZ adr	Null-Bit gesetzt
CC XXXX	
CNZ adr	Null-Bit zurückgesetzt
C4 XXXX	
CM adr	Vorzeichen-Bit gesetzt
FC XXXX	(Resultat negativ)
CP adr	Vorzeichen-Bit zurückgesetzt
F4 XXXX	(Ergebnis positiv)
CPE adr	Paritäts-Bit gesetzt
EC XXXX	
CPO adr	Paritäts-Bit zurückgesetzt
E4 XXXX	

Bedingter Rücksprung aus UP

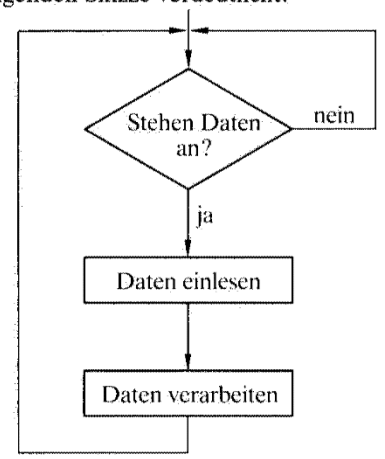
	kehre vom UP ins Hauptprogramm zurück, wenn ...
RC adr D8	Carry-Bit gesetzt
RNC D0	Carry-Bit zurückgesetzt
RZ adr C8	Null-Bit gesetzt
RNZ C0	Null-Bit zurückgesetzt
RM adr F8	Vorzeichen-Bit gesetzt (Resultat negativ)
RP adr F0	Vorzeichen-Bit zurückgesetzt (Ergebnis positiv)
RPE adr E8	Paritäts-Bit gesetzt
RPO adr E0	Paritäts-Bit zurückgesetzt

Noch einige Erläuterungen:

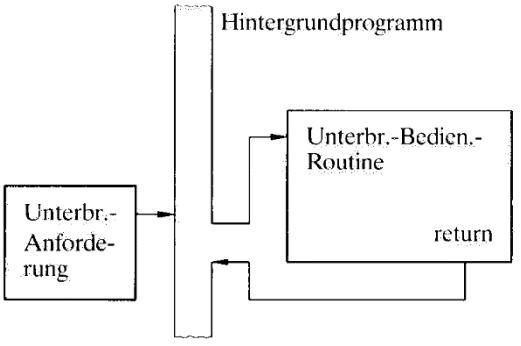
Zur Rettung der Rücksprung-Adresse benutzt der Prozessor, wie schon erwähnt, den Kellerspeicher (engl. stack = Stapel). Der Stack belegt meist die höchsten RAM-Adressen. Dabei zählt der Stackpointer SP immer mit, wenn im Stack etwas abgelegt wird (es werden stets 2-Byte-Wörter abgelegt). Dies geschieht folgendermaßen:
Der SP wird dekrementiert (SP−1), und sein Inhalt wird auf den Adreß-Bus gegeben zur Adressierung des Stack; hier wird das höherwertige Byte des abzuspeichernden 16-Bit-Wortes abgelegt. Dann wird der SP noch einmal dekrementiert (SP−2), und das niederwertige Byte wird abgelegt. Der SP zeigt nun auf das Byte, das gerade abgelegt wurde.
Der Kellerspeicher (Stack) enthält die Rückkehr-Adresse, weil bei der Ausführung eines CALL-Befehls nach Inkrementieren des Befehlszählers diejenige Adresse im Stack abgelegt wurde, die dann gerade im Befehlszähler (program counter, PC) stand.
Bei der Rückkehr ins Hauptprogramm, d. h. bei der Abarbeitung eines Return-Befehls, wird die gerettete Rückkehr-Adresse aus dem Kellerspeicher in den Befehlszähler PC zurückgebracht.

5.8.3 Programm-Unterbrechung

Der Mikroprozessor 8085 erlaubt eine hardwaremäßige Unterbrechung des laufenden Haupt- bzw. Hintergrund-Programms durch eigens dafür vorgesehene Anschlüsse am Prozessor-Baustein.
Die Unterbrechungs-Anforderung (interrupt request) erfolgt in der Regel von einem externen Gerät, das mit dem Prozessor Daten austauschen will.
Grundgedanke: Man könnte Auffang-Flip-Flops außerhalb des Prozessors aufbauen und diese per Programm zyklisch daraufhin abfragen, ob neue Daten zur Verarbeitung anstehen. Dies wird in der nachfolgenden Skizze verdeutlicht:



Da dieses Verfahren zwar einfach, aber zeitraubend ist, hat man die Unterbrechungs-Technik entwikkelt: Der Prozessor unterbricht beim Auftreten einer Unterbrechungs-Anforderung ein gerade laufendes Arbeits-(Hintergrund-)Programm, und es wird zu einer Unterbrechungsbedienungs-Routine (interrupt service routine) gesprungen. Dies zeigt die nachstehende Skizze.



Die Möglichkeit, den Prozessor in der Ausführung des gerade laufenden Programms zu unterbrechen, läßt sich softwaremäßig über ein Interrupt-Enable-Flip-Flop (INTE) im Prozessor steuern, und zwar mit Hilfe der Befehle EI (enable interrupt, Interrupt zulassen) und DI (disable interrupt, Interrupt verbieten).
Die Befehle EI und DI haben direkte Wirkung auf das Interrupt-Enable-Flip-Flop:

Die Befehle EI und DI:

EI	setzt INTE und gibt damit die Unterbrechung frei; wenn eine Unterbrechungs-Anforderung erfolgt, wird sie bearbeitet.
FB	Hex.-Code
DI	setzt INTE zurück und läßt dadurch keine Unterbrechung zu; wenn eine Unterbr.-Anforderung erfolgt, wird sie ignoriert.
F3	Hex.-Code

Zum Ablauf:

Erkennt der Prozessor eine Unterbrechung und ist sie per Programm zugelassen, so geschieht folgendes:
1. Der gerade bearbeitete Befehl wird beendet.
2. INTE wird zurückgesetzt.
3. Das unterbrechende Gerät liefert über die Hardware einen RST-Befehl, wenn die Unterbrechung über den Anschluß INTR erfolgte; der Prozessor führt den RST-Befehl aus. Dieser Befehl erscheint nicht im Speicher, und der Programmierer hat keine Kontrolle über ihn.

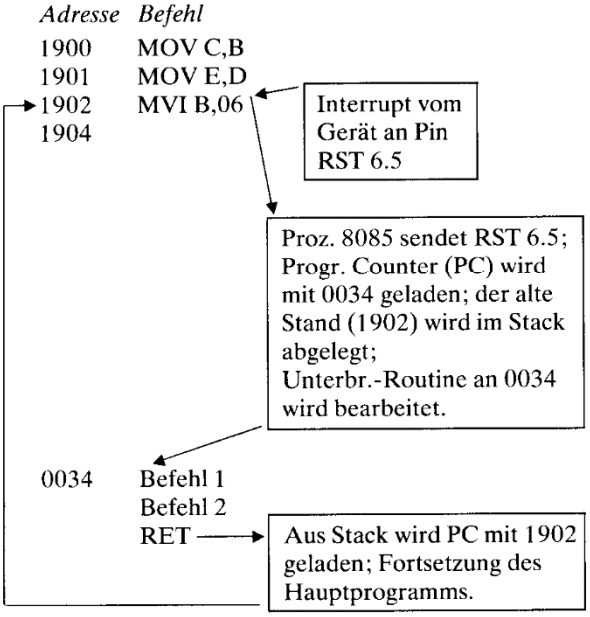
Wurde die Unterbrechung über einen der Anschlüsse TRAP, RST 7.5, RST 6.5 oder RST 5.5 (vgl. auch *Bild 3.1.1 und 3.1.2*) angefordert, dann verzweigt der Prozessor sofort auf die entsprechende Adresse (s. u.).
Liegen mehrere Unterbrechungs-Anforderungen gleichzeitig an, so werden sie nach der ihnen zugeordneten Priorität ausgeführt; die folgende Abbildung zeigt diese Prioritäten und die den verschiedenen Anschlüssen zugeordneten Verzweigungs-Adressen.

Unterbrechungs-System des 8085:

Priorität	Unterbr.-Anford. am Anschluß	RST-Befehl (hex.)	Softwarebeeinfluss.	Verzw.-Adresse (hex.)
niedrigste		RST 0 (C7)	EI (FB) bzw. DI (F3)	0000
		RST 1 (CF)		0008
		RST 2 (D7)		0010
		RST 3 (DF)		0018
	INTR	RST 4 (E7)		0020
		RST 5 (EF)		0028
		RST 6 (F7)		0030
		RST 7 (FF)		0038
	RST 5.5	einzeln		002C
	RST 6.5	maskierbar mit		0034
	RST 7.5	SIM-Befehl		003C
höchste	TRAP	nicht maskierbar		0024

Besonderheit:

Beim Anforderungs-Signal am Anschluß RST 7.5 wird die Unterbrechungs-Anforderung gespeichert. Wenn der Prozessor eine Unterbrechungs-Anforderung erkennt und bearbeitet, dann speichert er vor dem Verzweigen zur RST-Adresse die Absprung-Adresse des Hauptprogramms im Kellerspeicher. Wenn die Unterbrechungsbedienungs-Routine mit einem Return-Befehl endet, kann somit das Hauptprogramm anschließend an der Unterbrechungs-Stelle fortgesetzt werden.
Alle Anforderungs-Signale müssen bis zu ihrer Bedienung anliegen – mit Ausnahme des RST-7.5-Signals, das im RST-7.5-Flip-Flop gespeichert wird. Die nachfolgende Skizze zeigt ein Beispiel für einen Unterbrechungs-Lauf.



Für das Programmieren von Unterbrechungs-Routinen gilt allgemein: Jedes Register und jedes Bedingungs-Bit, das durch eine Unterbrechungs-Routine verändert wird, muß (mit PUSH) sichergestellt und vor der Rückkehr ins unterbrochene Programm (mit POP) wieder zurückgeladen werden.
Während der Unterbrechungs-Routine kann das INTE-Flip-Flop erneut mit dem Befehl EI freigegeben werden, so daß weitere Unterbrechungen erkannt werden können. Wird kein EI-Befehl gegeben, dann wird mit Ausnahme des TRAP-Interrupts keine Unterbrechung mehr angenommen.

Zwei Beispiele für eine Unterbrechungs-Routine:

1. Es wird eine neue Unterbrechung zugelassen:

PUSH B	B und C im Stack ablegen
PUSH PSW	A und F im Stack ablegen
EI	neue Unterbr.-Freigabe
•	Ausführung der Unterbr.-
•	Routine
•	
POP PSW	A und F zurückholen
POP B	B und C zurückholen
RET	Rückkehr ins Hauptprogramm

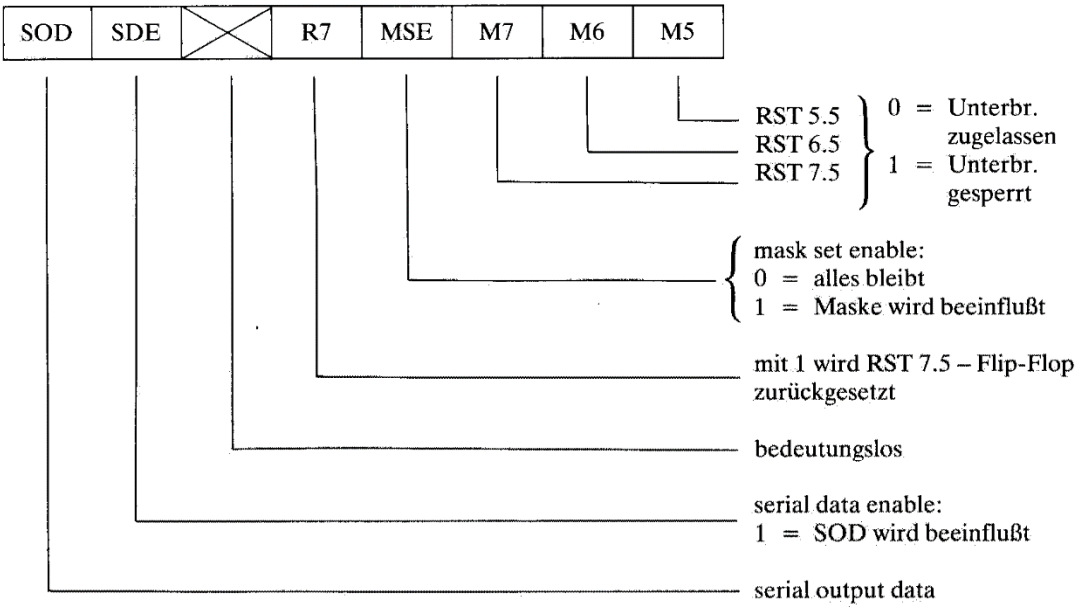
2. Es wird eine neue Unterbrechung erst am Ende der Unterbrechungs-Bedienungs-Routine zugelassen:

PUSH B	
PUSH PSW	sichern
•	Ablauf der Unterbr.-
•	Routine
•	
POP PSW	zurückholen
POP B	
EI	neue Unterbr. zulassen
RET	Rückkehr

Im zweiten Beispiel wird in jedem Falle noch der RET-Befehl ausgeführt, ehe eine neue Unterbrechung angenommen werden kann.

5.9 Die Befehle RIM und SIM

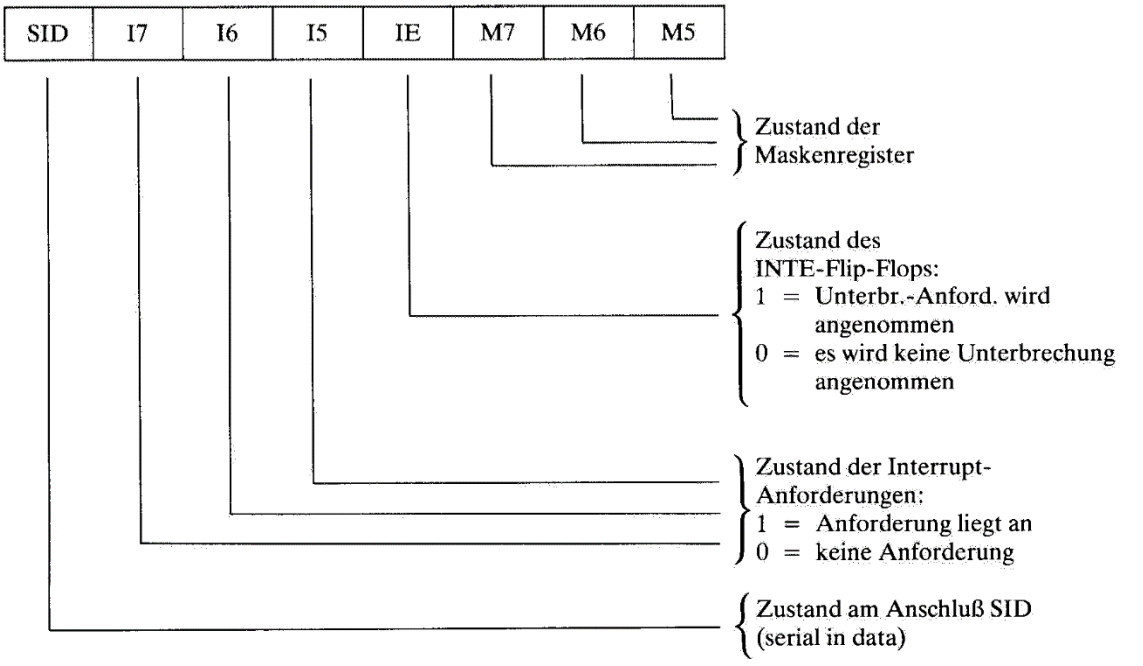
Diese beiden Befehle gibt es bei den Prozessoren 8080 und Z80 nicht; der 8085 benötigt sie zur Nutzung seiner zusätzlichen Eigenschaften.
Der Befehl SIM (= set interrupt mask) hat den Hex.-Code 30; er bewirkt folgendes:
a) Setzen der RST-5.5-, RST-6.5- bzw. RST-7.5-Masken über den Akkumulator;
b) gesondertes Rücksetzen des RST-7.5-Flip-Flops über den Akkumulator;
c) Ausgabe des Bits 2 des Akkus über den Anschluß SOD (serial out data) des 8085
Das im Akku vorliegende Bitmuster wird mit dem Befehl SIM wie folgt ausgewertet (dabei bedeutet Maske, daß die auf die drei niedrigtwertigen Bit gelegten Werte zusätzlich zu den Befehlen EI und DI darüber bestimmen, ob eine Unterbrechung zulässig ist):



Der Befehl RIM (read interrupt mask; Hex.-Code: 20) liest folgendes in den Akku ein:

a) den Zustand des Maskenregisters (RST 5.5, RST 6.5, RST 7.5);
b) die anstehenden Unterbrechungs-Anforderungen (RST 5.5, RST 6.5, RST 7.5);
c) den Zustand des Eingangs SID (dieser gelangt in das höchstwertige Bit des Akkus).

Das nach dem Befehl RIM im Akku befindliche Bitmuster wird wie folgt ausgewertet:



Im Rahmen dieser kurzen Einführung muß auf eine umfassende Information zur Behandlung der Unterbrechungs-Anforderungen verzichtet werden. In den Übungsbeispielen zur Reaktionszeit-Messung und zur ECB-Uhr werden Unterbrechungs-Anforderungen zugelassen und bearbeitet, so daß anhand dieser Beispiele die Anwendung leicht nachzuvollziehen ist. Weiteres kann der Leser im ausführlichen Datenbuch des Prozessors 8085 finden.

5.10 Der Befehl DAA

Der Befehl DAA (decimal adjust accumulator; Hex.-Code: 27) wird bei der Addition von Dezimalzahlen verwendet; er benötigt das Hilfscarry-Bit, das bei der vorangegangenen arithmetischen Operation gesetzt oder rückgesetzt wurde. Um das Hilfscarry-Bit richtig auswerten zu können, muß im Programm der Befehl DAA unmittelbar auf den betr. arithmetischen Befehl folgen.

Wirkung des Befehls DAA:

Er addiert zum Inhalt des Akkus den Wert sechs (binär 0110), wenn das Hilfscarry-Bit gesetzt oder wenn die Bit-Kombination im niederwertigen Halbbyte des Akkus größer als 9 ist. Zur Erklärung: Das Dezimalsystem benutzt nur 10 der 16 Bit-Kombinationen des Hexadezimalsystems, die durch vier Bit realisiert werden können; daher zur Korrektur, wenn erforderlich, die Addition von sechs.

Ein Beispiel:

Zur Zahl 23 (im BCD-Code: 0010 0011) soll eine 9 addiert werden. Das geschieht durch die Befehlsfolge:

```
MVI A, 23H
ADI 09
DAA
```

Dabei erhält man nach dem Befehl ADI 09 im Akku das Ergebnis 2CH:

```
  0010 0011   (23 im BCD-Code)
+ 0000 1001   (neun)
-------------------------------
  0010 1100 = 2C (im Hex.-Code).
```

Dies ist aber keine BCD-Zahl, da im niederwertigen Halbbyte ein Wert größer als 9 steht.

Der DAA-Befehl führt daher die Addition von 6 aus:

```
  0010 1100   (obiges Zwischenergebnis 2C)
+ 0000 0110   (sechs)
-------------------------------
  0011 0010 = 32 H
```

Somit erscheint im Akku nun die BCD-Zahl 32 (= 23 + 9).

5.11 Befehle, die nur das Carry-Bit bearbeiten

CMC (Hex.-Code: 3F) invertiert das Carry-Bit.
STC (Hex.-Code: 37) setzt das Carry-Bit auf 1.

5.12 Die Steuerbefehle HLT und NOP

Wenn der Mikroprozessor einen Haltbefehl (NOP, Hex.-Code: 76) bearbeitet, dann bleibt er im Haltzustand, bis eine Interrupt-Anforderung erfolgt. Wurde eine solche nicht zugelassen, so kann der Prozessor nur durch RESET aus dem Haltzustand geholt werden. Das hat aber zur Folge, daß die anwenderspezifische Initialisierung z. B. der E/A-Bausteine verlorengeht.
Der NOP-Befehl (Hex.-Code: 00) dient als Platzhalter im Programmspeicher oder zur Erzeugung von Zeiten. Der Prozessor decodiert den Befehl NOP und bearbeitet dann den nächsten; es vergeht also nur Zeit, und sonst geschieht nichts.

6. Die Handhabung des Experimentier-Computers ECB 85

6.1 Der Aufbau des ECB 85

Bevor in *Teil 2* einige teilweise umfangreiche Versuche mit dem Experimentier-Computer ECB 85 durchgeführt werden, sei an dieser Stelle das Gerät mit seiner Handhabung beschrieben. Die wichtigsten Funktionseinheiten des ECB 85 wurden schon in *Kapitel 3* vorgestellt. Besonders wichtig für den Anwender sind dabei die *Bilder 3.2.1 und 3.3.5* (Speicher- und E/A-Adreßraum) sowie *Bild 3.4.1* (Anwender-Anschlüsse an der 96poligen Steckerleiste). Neben den Listen der 8085-Befehle *(S. 140ff.)* sollten diese Abbildungen beim Erstellen von Programmen und beim Experimentieren mit Peripherie-Schaltungen stets zur Hand sein.

Bild 6.1.1 zeigt in einer vereinfachten zeichnerischen Darstellung die Lage der wichtigsten Funktionseinheiten und Bedienungselemente auf der ECB-85-Platine. Neben dem Lochraster-Experimentierfeld sind zu erkennen:

- der Stecksockel für das Anwender-EPROM
- die Programmierlampe L
- die beiden Schalter S1 und S2
- die Kassettenrecorder-Anschlußbuchse CR
- das 8stellige Anzeigefeld (7-Segment-Anzeigen)
- die Tastatur.

Je nach Bedarf ist der EPROM-Stecksockel vor Inbetriebnahme des Geräts zu bestücken oder freizulassen; während des Betriebs muß beim Entfernen oder Aufstecken eines EPROM-Bausteins die RESET-Taste gedrückt sein.

Schalter	Stellung	Funktion
S1 (Betriebsart)	2	EPROM lesen
	1 PRGR	EPROM schreiben (programmieren)
S2 (Sprungziel nach RESET)	2 MON	Monitor-Programm (Adr. 0000)
	1 ANW	Anwender-Programm (Adr. 0800) (EPROM)

Bild 6.1.2: Die Bedeutung der beiden Schiebeschalter S1 und S2 auf der ECB-85-Platine.

Die Schiebeschalter S1 und S2 haben die in *Bild 6.1.2* aufgeführte Bedeutung. Beim Eingeben und Testen von Programmen müssen beide Schalter in Stellung 2 stehen; es wird also nach RESET (ebenso beim Einschalten des Geräts) auf die Adresse 0000 des Monitor-Programms gesprungen (das damit gestartet wird), und ein evtl. auf dem Stecksockel befindliches EPROM kann gelesen werden.

6.1.1 Das Anzeigefeld

Nach Anlegen der Versorgungsspannung (+5 V) blinkt der Cursor, in diesem Falle das unterste waagerechte Segment der linken Stelle; der rechts daneben liegende Dezimalpunkt leuchtet dauernd: –. Dieselbe Anzeige erhält man nach Drücken der RESET-Taste (s. u.). Nun kann ein Kommando-Symbol eingegeben werden.

Je nachdem, welches Kommando gerade bearbeitet wird *(Bild 6.1.3)*, hat das Anzeigefeld eine der drei in *Bild 6.1.4* gezeigten Strukturen. Der Cursor wird dadurch realisiert, daß diejenige Anzeigestelle blinkt, in der gerade eine Eingabe erwartet wird. Zu Beginn jedes Kommando-Schrittes zeigt der ECB 85 in jeder Anzeigestelle einen sogen. Vorschlagswert

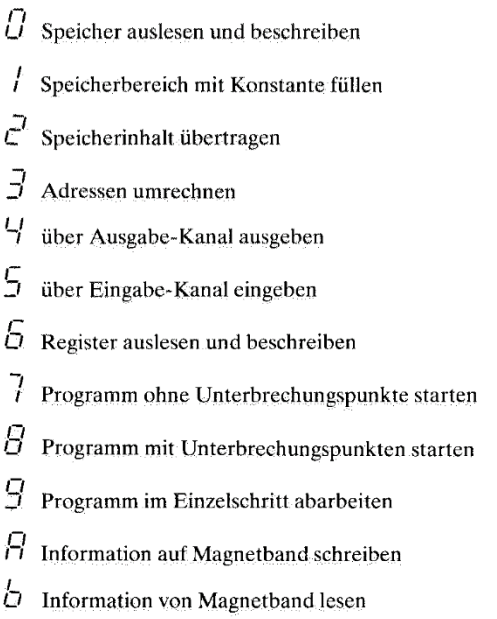


Bild 6.1.3: Die Symbole im Kommandofeld und ihre Funktion.

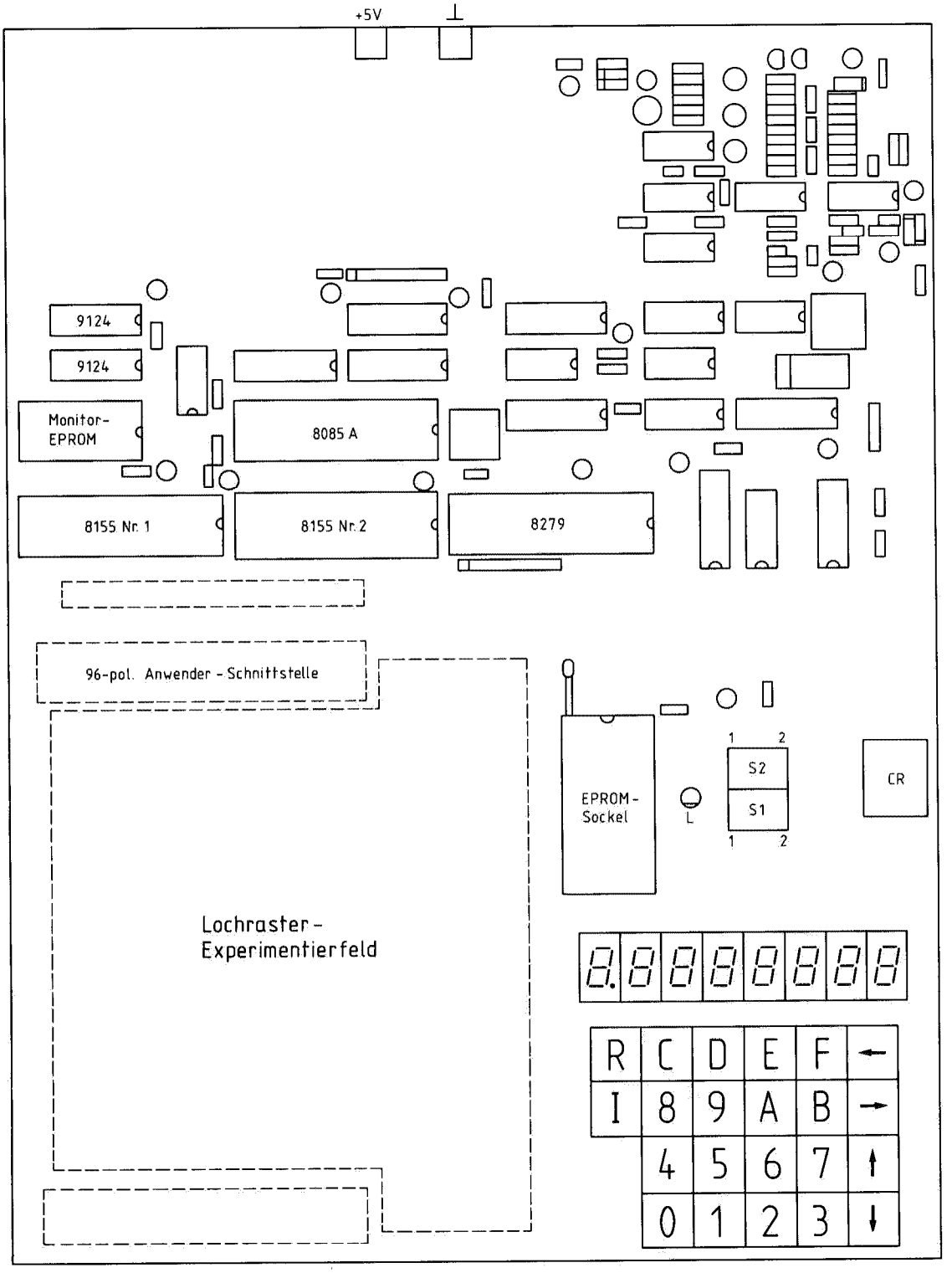


Bild 6.1.1: Die Bestückung der ECB-85-Platine. Bezeichnet sind nur die wichtigsten ICs (siehe *Kapitel 3*). Die neben dem Lochrasterfeld befindlichen Bedienungselemente sind Gegenstand dieses Kapitels 6.1.

an, der vom Benutzer durch einfaches Quittieren übernommen oder durch eine Eingabe an der betreffenden Stelle geändert werden kann. Dies wird bei der Beschreibung der einzelnen Kommandos in *Kapitel 6.2* deutlich werden. Weiterhin ist anzumerken, daß alle Adressen und Daten stets im Hexadezimal-Code angezeigt werden bzw. einzugeben sind.

Das Kommandofeld

Die linke Anzeigestelle dient ausschließlich dazu, das gerade bearbeitete Kommando anzuzeigen; der rechts davon liegende Punkt leuchtet immer. Die 12 möglichen Kommando-Symbole des ECB 85 sind in *Bild 6.1.3* gezeigt.

Das Adreßfeld

Im Falle des Kommandos 0 (Speicher lesen und beschreiben) zeigt das Adreßfeld den Hex.-Code der 4stelligen Adresse desjenigen Speicherplatzes an, der gerade angesprochen wird; die Anzeige sieht dann aus wie in *Bild 6.1.4a)* dargestellt. Beim Ausführen anderer Kommandos umfaßt das Adreßfeld nur zwei Stellen und zeigt dann entweder einen Parameter-Namen, eine Port-Adresse oder einen Registerpaar-Namen; siehe *Bild 6.1.4b) und c)*. Die möglichen Parameter- und Registerpaar-Namen sind in *Bild 6.1.5* zusammengestellt.

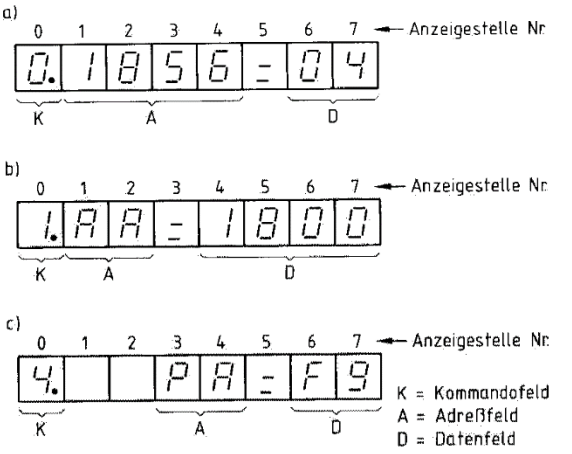


Bild 6.1.4: Drei Beispiele für die 8stellige Anzeige des ECB 85.
a) Mit Kommando 0 wird die Speicherzelle mit der Adresse 1856 (Hex.-Code!) gelesen bzw. das Datenbyte 04 eingeschrieben.
b) Mit Kommando 1 soll ab der Anfangs-Adresse (AA) 1800 eine Konstante eingegeben werden.
c) Mit Kommando 4 soll über den Ausgabekanal mit der Peripherie-(Port-)Adresse (PA) F9 ausgegeben werden.
Die Kommando-Abläufe werden in *Kapitel 6.2* eingehend beschrieben; an dieser Stelle soll nur die Struktur des Anzeigefeldes gezeigt werden.

a) Parameternamen:

AA	Anfangsadresse
EA	Endadresse
bA	Bestimmungsadresse
SA	Startadresse
H 1	Haltadresse 1
H2	Haltadresse 2
AU	Anfangsadresse des umzurechnenden Bereichs
EU	Endadresse des umzurechnenden Bereichs
PA	Peripherie-Adresse
db	Datenbyte

b) Registerpaare:

AF	Akkumulator und Zustandsbyte (Flagbyte)
bC	Registerpaar B
dE	Registerpaar D
HL	Registerpaar H
PC	Programmzähler (program counter)
SP	Kellerspeicher-Zeiger (stack pointer)
IS	Interrupt-Status

Bild 6.1.5: Die möglichen Symbole im Adreßfeld und ihre Bedeutung.

Das Datenfeld

Hier steht entweder der Inhalt einer Speicherzelle (1 Daten-Byte, 2stellige Hex.-Anzeige) oder das an eine Port-Adresse auszugebende bzw. von dort einzulesende Daten-Byte; siehe *Bild 6.1.4a) und c)*. Wenn im Adreßfeld jedoch bestimmte Parameter wie Speicher-Adressen oder Registerpaare angesprochen werden, dann umfaßt das Datenfeld vier Hex.-Stellen (2 Bytes); siehe *Bild 6.1.4b)*.

6.1.2 Die Tastatur

Zum Eingeben von Kommandos, Daten, Adressen usw. besitzt der ECB 85 insgesamt 22 Tasten (siehe *Bild 6.1.1*), und zwar die beiden Hardware-Tasten R und I sowie 20 Software-Tasten.
Die *R-Taste* (RESET-Taste) dient zum Rücksetzen des ECB 85. Steht dabei der Schalter S2 auf Pos. 2 (MON, siehe *Bild 6.1.2*), so bewirkt das Betätigen der R-Taste den Start des Monitor-Programms bei der Adresse 0000. Die R-Taste wird hauptsächlich dazu benutzt, den Prozessor aus einem unerwünschten oder unübersichtlichen Zustand wieder in den definierten Ausgangszustand zu bringen. Die *I-Taste* (Interrupt-Taste) dient zum Auslösen eines Interrupts RST 7.5 (Adresse 003C). Vgl. hierzu *Kapitel 5.9.*
Bei den Software-Tasten sind zwei Gruppen zu unterscheiden:

- *16 Hexadezimaleingabe-Tasten:* 0...9 und A...F, angeordnet in einen 4×4-Feld; mit ihnen werden Adressen, Daten und Kommandos eingegeben. Es wurde schon erwähnt, daß alle Eingaben und Anzeigen am ECB 85 im Hex.-Code erfolgen.
- Vier *Steuer- bzw. Abschluß-Tasten* (rechte Spalte des Tastenfeldes). Die beiden *Abschlußtasten* ↓ und ↑ dienen vor allem zum Abschließen bei der Eingabe von Kommando-Symbolen und Parametern. Oft haben sie auch noch Nebenwirkungen wie etwa das sukzessive Anwählen aufeinanderfolgender Registerpaare oder das automatische Inkrementieren (↓) bzw. Dekrementieren (↑) von Speicher-Adressen bei der Eingabe von Daten in das RAM. Näheres hierzu wird bei der Behandlung der Kommandos in *Kapitel 6.2* erläutert werden. Mit den beiden Steuertasten → und ← kann man den Cursor (die Schreibmarke) verschieben, so daß der Benutzer bestimmen kann, ob er im Kommando-, im Adreß- oder im Datenfeld eingeben will.

Kommando-Symbol	Funktion	Funktionsgruppe
0	Speicher auslesen und beschreiben	Kommandos mit Bezug auf den Speicher
1	Speicherbereich mit Konstante füllen	
2	Speicherinhalt übertragen	
3	Adressen umrechnen	
4	über Ausgabe-Kanal ausgeben	Kommandos zum Testen der Hardware
5	über Eingabe-Kanal eingeben	
6	Register auslesen und beschreiben	Kommandos zum Testen der Software
7	Programm ohne Unterbrechungspunkte starten	
8	Programm mit Unterbrechungspunkten starten	
9	Programm im Einzelschritt abarbeiten	
A	Information auf Magnetband schreiben	Kommandos mit Bezug auf das Magnetbandgerät
B	Information von Magnetband lesen	

Bild 6.2.1: Die Monitor-Kommandos des ECB 85.

6.2 Die Monitor-Kommandos

Wie in *Kapitel 3.2* schon erwähnt, ermöglicht das Monitor-Programm (beim ECB 85 im Adreßbereich 0000–07FF) dem Benutzer die Kommunikation mit dem Prozessor über Tastatur und Anzeigefeld. Mit anderen Worten: der Benutzer kann mit Hilfe des Monitor-Programms Befehle, Adressen und Daten eingeben sowie Programme starten oder Informationen übertragen. In *Bild 6.2.1* sind die 12 Monitor-Kommandos des ECB 85 nach Funktionsgruppen geordnet aufgeführt.
Im folgenden werden die Kommandos und ihre Wirkungsweise eingehend besprochen. Dazu noch eine Vorbemerkung: In den *Bildern 6.2.2–6.2.12* wird der genaue Ablauf der Kommandos dargestellt. Dabei werden mit dem Symbol X Zufalls- oder Vorschlagswerte bezeichnet, die das Monitor-Programm des ECB 85 beim Aufruf des jeweiligen Kommandoparameters vorgibt. Durch einfaches Quittieren oder Verschieben des Cursors kann dieser Wert übernommen werden. In den Abbildungen wird jedoch immer die komplette Neueingabe gezeigt. Die Position des Cursors (der Schreibmarke) ist in den erwähnten Bildern stets durch ein Dreieck unter der betr. Anzeigestelle kenntlich gemacht.

Kommando 0: Speicher lesen und beschreiben (Bild 6.2.2)

Steht der Cursor im Adreßfeld oder hat man ihn mit einer der Positions-Tasten → und ← dorthin gebracht, so kann man jede der 65536 Adressen (64 K, siehe *Kapitel 3.2*) eingeben, deren Inhalt dann sogleich im Datenfeld (Anzeigestellen Nr. 6 und 7, siehe *Bild 6.1.4a)* angezeigt wird. Ist die Adresse komplett eingegeben, dann springt der Cursor automatisch in das Datenfeld. Das Betätigen einer der 16 Hex.-Tasten wird jetzt vom Monitor-Programm als Überschreiben (Ändern) des Inhalts der angezeigten

Schritt Nr.	Bedeutung	Betätigung	Anzeige	Quittung	Anzeige nach Quittieren
1	Start Kommando 0	Cursor nach links, Hex.-Taste 0	0. ▲	↓	0. XXXX=YY ▲
2	Eingabe der Adresse	4 Hex.-Tasten, z. B. 1,8,0,0	0. 1800=YY ▲		
	Eingabe eines Datenbytes in Adr. 1800	2 Hex.-Tasten, z. B. 3, E	0. 1800=3E ▲	↓	0. 1801=YY ▲
3	Eingabe eines Datenbytes in Adr. 1801	2 Hex.-Tasten, z. B. C, D	0. 1801=Cd ▲	↓	0. 1802=YY ▲
4	Eingabe eines Datenbytes in Adr. 1802	2 Hex.-Tasten, z. B. 7, 6	0. 1802=76 ▲	↓	0. 1803=YY ▲
	⋮ entweder so weiter, *oder* z. B.:				
–	Kontrolle, ob Daten wie gewünscht vorhanden:			↑	0. 1802=76 ▲
				↑	0. 1801=Cd ▲

Bild 6.2.2: Kommando 0, Speicher auslesen und beschreiben.

Speicheradresse interpretiert; vgl. aber die unten erwähnten Besonderheiten. Gibt man mehr als zwei Zeichen im Datenfeld ein, dann wird mit jedem weiteren Drücken einer Hex.-Taste stets die Stelle Nr. 7 überschrieben. Erst das Betätigen einer der Abschlußtasten beendet das Beschreiben der angezeigten Speicheradresse. Dabei ist zu beachten: Das Betätigen einer der Abschlußtasten ↓ und ↑ bewirkt in jedem Falle, daß die Dateneingabe beendet wird (es ist hierbei gleichgültig, wo sich der Cursor im Adreß- oder Datenfeld befindet) und daß die Speicheradresse inkrementiert bzw. dekrementiert, also die nächste Adresse angezeigt, wird.

Besonderheiten:

1. Wird eine Adresse angewählt, für die kein Baustein vorhanden ist (d. h. für die der Speicherraum nicht belegt ist), so wird als Inhalt das niederwertige Byte der Adresse angezeigt.
2. Daten im EPROM-Bereich können zwar gelesen, aber nicht geändert (überschrieben) werden – es sei denn, das Anwender-EPROM ist durch den Schalter S1 in Stellung 1 (PRGR) zum Schreiben freigegeben. Versucht man dennoch, im EPROM-Bereich das Datenbyte durch Eingeben anderer Hex.-Werte zu ändern, so reagiert die Anzeige nicht, und das Datenbyte bleibt unverändert. Das hat seinen Grund in folgendem: Bei jeder Eingabe eines Hex.-Zeichens im Datenfeld schreibt das Monitor-Programm diesen Wert in die betr. Speicherzelle und liest deren Inhalt sofort wieder aus, der dann in der Anzeige erscheint; im EPROM-Bereich ist das also der alte, unveränderte Inhalt. Dieses vom Monitor-Programm automatisch durchgeführte Kontroll-Lesen hat noch einen weiteren Vorteil: Hat der Benutzer mit dem Kommando 0 ein Datenbyte in den RAM-Speicher eingegeben und erscheint dieses in der Anzeige, so kann er sicher sein, daß es auch wirklich in den Speicher übernommen wurde.
3. Das Beenden des Kommandos 0 ist möglich, indem man den Cursor mit ← oder → in das Kommandofeld bringt und ein neues Kommandosymbol eingibt.

Kommando 1: Speicherbereich mit einer Konstanten füllen (Bild 6.2.3)

Mit diesem Kommando kann ein Speicherbereich von der Anfangs-Adresse (AA) bis zur End-Adresse

Schritt Nr.	Bedeutung	Betätigung	Anzeige	Quittung	Anzeige nach Quittieren
1	Start Kommando 1	Cursor nach links, Hex.-Taste 1	1. ▲	↓	1AA = XXXX ▲
2	Eingabe der Anfangs-Adresse (AA)	4 Hex.-Tasten, z. B. 1,8,0,0	1AA = 1800 ▲	↓	1EA = XXXX ▲
3	Eingabe der End-Adresse (EA)	4 Hex.-Tasten, z. B. 1,8,1,0	1EA = 18 10 ▲	↓	1 db = XX ▲
4	Eingabe der Konstanten (des Datenbytes, Symbol db)	2 Hex.-Tasten, z. B. E,F	1 db = EF ▲	↓	1. ▲ (Kommando ist ausgeführt)

Bild 6.2.3: Kommando 1, Speicherbereich mit einer Konstanten füllen.

(EA) mit ein und demselben Datenbyte (d. h. mit einer Konstanten) beschrieben werden. Dies wird oft das Datenbyte 00 H sein (Op.-Code für den 8085-Befehl NOP); z. B. kann so ein Vorspann für das Überspielen auf Magnetband (Kassette) erzeugt werden, oder man realisiert in einem Programm damit Platzhalter. Das beim Kommando 0 beschriebene Kontroll-Lesen der Speicherzellen-Inhalte findet auch beim Kommando 1 statt. Wenn z. B. versucht wird, einen EPROM-Bereich mit einer Konstanten zu füllen, so ist das aus dem Speicher ausgelesene Datenbyte nicht mit der Konstanten identisch; es ist also ein Fehler aufgetreten. In diesem Falle erscheint in der Anzeige die Adresse dieser Speicherzelle mit ihrem Inhalt, und die Ausführung des Kommandos 1 wird abgebrochen. In diesem Zustand (der Cursor ist nicht vorhanden) kann das Kommando 1 nur mit der R-Taste beendet werden. Der Benutzer kann nach Ausführung des Kommandos 1 zusätzlich – zumindest stichprobenweise – mit dem Kommando 0 nachprüfen, ob die Konstante in den geforderten Speicherbereich übernommen wurde.

Kommando 2: Speicherinhalt übertragen (Bild 6.2.4)

Dieses Kommando kann man auch mit „Speicherbereich verschieben" bezeichnen: Alle Inhalte der Speicherzellen von der Anfangs-Adresse (AA) bis zur End-Adresse (EA) werden in den mit der Ziel-(Bestimmungs-)Adresse (BA) beginnenden Speicherbereich verschoben. Dabei dürfen sich die beiden Bereiche überschneiden (Bestimmungs-Adr. ist kleiner als Anfangs- oder End-Adr. bzw. sie liegt zwischen Anfangs- und End-Adr.); auch in diesem Falle führt das Monitor-Programm des ECB 85 dieses Kommando 2 so aus, daß keine Informationen durch Überschreiben zum falschen Zeitpunkt vernichtet werden. Auch hier findet ein automatisches Kontroll-Lesen statt. Tritt ein Fehler auf, so reagiert das System ebenso wie beim Kommando 1 beschrieben.

Anwendungsmöglichkeiten des Kommandos 2: Es können Speicherplätze innerhalb eines Programms freigesetzt werden zum Einfügen zusätzlicher Befehle, oder es können Programme vom RAM in das Anwender-EPROM übertragen werden.

Schritt Nr.	Bedeutung	Betätigung	Anzeige	Quittung	Anzeige nach Quittieren
1	Start Kommando 2	Cursor nach links, Hex.-Taste 2	2. ▲	↓	2.AA = XXXX ▲
2	Eingabe der Anfangs-Adresse (AA)	4 Hex.-Tasten, z. B. 0,0,0,0	2.AA = 0000 ▲	↓	2.EA = XXXX ▲
3	Eingabe der End-Adresse (EA)	4 Hex.-Tasten, z. B. 0,0,4,0	2.EA = 0040 ▲	↓	2.bA = XXXX ▲
4	Eingabe der Bestimmungs-Adresse (bA) (auch: Ziel-Adresse)	4 Hex.-Tasten, z. B. 1,8,0,0	2.bA = 1800 ▲	↓	2. ▲ (Kommando ausgeführt)

Bild 6.2.4: Kommando 2, Speicherinhalt übertragen.

Kommando 3: Adressen umrechnen (Bild 6.2.5)

Mit diesem Kommando werden Sprungadressen in Programmen umgerechnet, wenn die Sprungbefehle in einem vorzugebenden Adreßbereich liegen und die Sprungadressen in einen auch vorzugebenden Bereich zeigen. Im allgemeinen wird das Kommando 3 im Anschluß an Kommando 2 verwendet, d. h. wenn zuvor ein Programm(teil) in einen anderen Speicherbereich verschoben wurde und nun die Sprungadressen noch mit dem alten Speicherbereich verknüpft sind. Zu dem in *Bild 6.2.5* gezeigten Beispiel: Hier wurde das Programm zuvor mit dem Kommando 2 vom Speicherbereich 1920–1936 in den Bereich 1926–193C verschoben. Daher ist AA = 1920 (Anfangs-Adresse des Ursprungsbereichs) und EA = 1936 (End-Adresse des Ursprungsbereichs). Verschoben wurde zur Bestimmungs-Adresse (BA) 1926; diese Adresse ist also normalerweise auch der Anfang des umzurechnenden Bereichs (AU), der bei 193C endet (EU). Mit anderen Worten: In dem Bereich zwischen AU und EU, in dem sich das Programm nach der Verschiebung befindet, werden alle Adressen umgerechnet, die in den Ursprungsbereich zwischen AA und EA zeigen. Dieses Umrechnen geschieht so, daß die Sprungbefehle danach in den bei BA beginnenden Bereich zeigen.

Besonderheiten:

1. Es werden nur Adressen in bedingten oder unbedingten Sprungbefehlen umgerechnet (z. B. JMP, CALL, RET), aber nicht in Befehlen wie beispielsweise LXI H,1256 H; denn hier treten die Zahlen zunächst als Daten auf und werden erst später, je nach Programmteil, als Adressen interpretiert. Der

Schritt Nr.	Bedeutung	Betätigung	Anzeige	Quittung	Anzeige nach Quittieren
1	Start Kommando 3	Cursor nach links, Hex.-Taste 3	3. ▲	↓	3.AA= XXXX ▲
2	Eingabe der alten Anfangs-Adresse (AA)	4 Hex.-Tasten, z. B. 1,9,2,0	3.AA= 1920 ▲	↓	3.EA= XXXX ▲
3	Eingabe der alten End-Adresse (EA)	4 Hex.-Tasten, z. B. 1,9,3,6	3.EA= 1936 ▲	↓	3.BA= XXXX ▲
4	Eingabe der neuen Anfangs-Adresse (Bestimmungs-Adr.: bA)	4 Hex.-Tasten, z. B. 1,9,2,6	3.BA= 1926 ▲	↓	3.AU= XXXX ▲
5	Eingabe des Anfangs des umzurechnenden Bereichs (AU), bzw. der neuen Anf.-Adr.	4 Hex.-Tasten, z. B. 1,9,2,6	3.AU= 1926 ▲	↓	3.EU= XXXX ▲
6	Eingabe des Endes des umzurechnenden Bereichs (EU), bzw. der neuen End-Adr.	4 Hex.-Tasten, z. B. 1,9,3,C	3.EU= 193C ▲	↓	3. ▲ (Kommando ausgeführt)

Bild 6.2.5: Kommando 3, Adressen umrechnen.

genannte Befehl steht also auch nach dem Umrechnen mit Kommando 3 als 21 56 12 im Speicher.
2. Die Anfangs-Adresse des umzurechnenden Befehls muß auf einen Op.-Code zeigen.
3. Kontroll-Lesen und Fehlermeldung wie bei Kommando 1 und 2 erfolgen hier nicht.

Kommando 4: Über Ausgabe-Kanal ausgeben (Bild 6.2.6)

Mit dem Kommando 4 können Informationen an die E/A-Ports ausgegeben werden. Zuvor müssen die Ports des anzusprechenden Baustein 8155 (Nr. 1 oder Nr. 2, vgl. *Kapitel 3.3*) zur Ausgabe programmiert worden sein. Dies kann auch mit Hilfe des Kommandos 4 selbst geschehen, indem zunächst das entsprechende Steuerwort als „Datenbyte" an das Steuerwort-Register des Bausteins 8155 ausgegeben wird und danach das eigentliche Datenbyte an den mit dem Steuerwort zur Ausgabe programmierten E/A-Port. Das ist in *Bild 6.2.6* für folgenden Fall gezeigt: Alle Ports (Kanäle) des 8155 Nr. 2 werden zur Ausgabe programmiert (Steuerwort 0F an das Steuerwortregister mit der Adresse F0 ausgeben, vgl. *Bild 3.3.5 und 3.3.6*), und dann wird das Datenwort 75 an Port A des 8155 Nr. 2 (E/A-Adr. F1) ausgegeben.

Besonderheiten:

1. Das Monitor-Programm prüft nicht, ob unter der angegebenen Adresse auch tatsächlich ein Ausgabe-Kanal vorhanden ist. Eine Ausgabe an nicht vorhandene Kanäle hat keine Auswirkungen und ist für den Benutzer auch nicht erkennbar.
2. Das Beenden des Kommandos 4 ist möglich, indem man den Cursor in das Kommandofeld bringt und ein neues Kommandosymbol eingibt.

Schritt Nr.	Bedeutung	Betätigung	Anzeige	Quittung	Anzeige nach Quittieren
1	Start Kommando 4	Cursor nach links, Hex.-Taste 4	4. ▲	↓	4. PA=XX ▲
2	Eingabe der Peripherie-Adresse (PA)	2 Hex.-Tasten, z. B. F,0	4. PA=F0 ▲	↓	4. F0=XX ▲
3	Eingabe des Datenbytes (Steuerworts)	2 Hex.-Tasten, z. B. 0, F	4. F0=0F ▲	↓	4. F0=0F ▲
4	Cursor ins Adreßfeld bringen	← (1 mal)	4. F0=0F ▲		
5	Eingabe der Peripherie-Adresse (F1) von Port A	1 Hex.-Taste z. B. 1	4. F1=0F ▲		
6	Eingabe des Datenbytes	2 Hex.-Tasten, z. B. 7,5	4. F1=75 ▲	↓	4. F1=75 ▲ (Datenbyte wurde ausgegeben)

Bild 6.2.6: Kommando 4, über Ausgabe-Kanal ausgeben.

Kommando 5: Über Eingabe-Kanal eingeben (Bild 6.2.7)

Dieses Kommando dient der Eingabe von Daten über die E/A-Ports der Bausteine 8155 Nr. 1 und Nr. 2 des ECB 85. Das über die Programmierung der Ports bei Kommando 4 Gesagte gilt hier entsprechend; allerdings sind nach Betätigung der RESET-Taste alle Ports des 8155 zur Eingabe programmiert, so daß von hier ohne besondere Vorbereitung eingelesen werden kann. Die bei Kommando 4 aufgeführten Besonderheiten gelten hier entsprechend (auf Eingabe übertragen). Wenn die Peripherie-Anschlüsse offen sind, läßt sich über das zu erwartende Datenbyte nichts voraussagen.

Kommando 6: Register auslesen und beschreiben (Bild 6.2.8)

Mit diesem Kommando (das nur auf dieselbe Weise wie z. B. Kommando 0 beendet werden kann) können die Inhalte der Registerpaare der CPU angezeigt bzw. verändert werden. Das Betätigen einer der Abschlußtasten bringt das nächste Registerpaar zur Anzeige. Dabei gelten die Anzeigestellen Nr. 4 und 5 (vgl. *Bild 6.1.4*) für den in Stelle Nr. 1 angezeigten Teil des Registerpaares. Beispiel: Im Anzeigefeld steht 6.AF=5738. Dann ist der Akku-Inhalt 57 und der Inhalt des Flagbytes 38 (jeweils wie immer im Hex.-Code!).

Schritt Nr.	Bedeutung	Betätigung	Anzeige	Quittung	Anzeige nach Quittieren
1	Start Kommando 5	Cursor nach links, Hex.-Taste 5	5. ▲	↓	5. PA = XX ▲
2	Eingabe der Peripherie-Adresse (PA)	2 Hex.-Tasten, z. B. F,9	5. PA = F9 ▲	↓	5. F9 = YY ▲ (Kommando ausgeführt, YY ist das eingelesene Byte; vgl. Text)

Bild 6.2.7: Kommando 5, über Eingabe-Kanal eingeben.

Schritt Nr.	Bedeutung	Betätigung	Anzeige	Quittung	Anzeige nach Quittieren
1	Start Kommando 6	Cursor nach links, Hex.-Taste 6	6. ▲	↓	z.B.: 6.PC = XXXX ▲
2	Eingabe von 2 Bytes in das angezeigte Registerpaar, hier in den Program Counter (PC)	4 Hex.-Tasten, z.B. 1,8,0,0	6.PC = 1800 ▲	↓	6.SP = XXXX ▲
3	Eingabe von 2 Bytes in den Stack Pointer (SP) . . . usw.	4 Hex.-Tasten, z.B. 1,C,0,0	6.SP = 1C00 ▲	↓	6. 15 = XXXX ▲

Reihenfolge der angezeigten Registerpaare mit der Abschlußtaste ↓:

PC (Program Counter)
SP (Stack Pointer)
IS (Interrupt Status)
AF (Akku u. Flagbyte)
BC (Reg.-Paar BC)
DE (Reg.-Paar DE)
HL (Reg.-Paar HL)
PC
.
.
.
usw.

Mit der Abschlußtaste ↑: entgegengesetzte Reihenfolge

Bild 6.2.8: Kommando 6, Register auslesen und beschreiben.

Kommando 7: Programm ohne Unterbrechungs-Punkte starten (Bild 6.2.9)

Nach Eingabe des Kommandos 7 läuft das gestartete Programm in Echtzeit ab. Vom Benutzer ist dazu nur die Start-Adresse einzugeben und eine der Abschlußtasten zu drücken (siehe *Bild 6.2.9*). Der letzte Befehl eines Anwenderprogramms soll stets ein Haltbefehl (HLT, Hex.-Code 76) sein. Den Programmdurchlauf bis zum Halt erkennt man daran, daß keine Ziffer mehr blinkt, d. h. der Cursor verschwindet, und daß man nur mit RESET in das Monitor-Programm (zur Eingabe von Kommandos) zurückkehren kann. Eine Rückkehr in das Monitor-Programm ist auch mit der Taste I (Interrupt) möglich, wenn vor dem Halt-Befehl die Unterbrechung am RST-7.5-Eingang freigegeben wurde. Hierbei wird dann der aktuelle PC-Stand (Progr.-Zähler) auf der Anzeige ausgegeben.

Schritt Nr.	Bedeutung	Betätigung	Anzeige	Quittung	Anzeige nach Quittieren
1	Start Kommando 7	Cursor nach links, Hex.-Taste 7	7. ▲	↓	7.SA= XXXX ▲
2	Eingabe der Start-Adresse (SA)	4 Hex.-Tasten, z. B. 1,8,0,0	7.SA= 1800 ▲	↓	7.SA= 1800 ▲ (Programm wird ausgeführt)
3	Unterbrechung durch RST 7.5	Taste I	7.PC= 18 10 ▲ (z. B.)		

Bild 6.2.9: Kommando 7, Programm ohne Unterbrechungs-Punkte starten.

Kommando 8: Programm mit Unterbrechungspunkten starten (Bild 6.2.10)

Kommando 8 startet das Programm und erlaubt zuvor die Eingabe von zwei Unterbrechungs-Punkten (Halt-Adressen). Nach Programmstart (Taste ↓ im Schritt 4 drücken, siehe *Bild 6.2.10*) läuft die Anzeige der Programm-Adressen im Datenfeld durch und bleibt bei Erreichen einer der Halt-Adressen stehen. Der Cursor steht dann im Kommandofeld. Nun kann das Programm mit Kommando 8 fortgesetzt werden, wobei zwei neue Haltepunkte eingegeben werden können. Das Programm läuft mit Kommando 8 ca. 20mal langsamer ab als in Echtzeit mit Kommando 7. Die Tastatur und die Anzeige des ECB 85 können vom Anwenderprogramm im Kommando 8 nicht benutzt werden.

Schritt Nr.	Bedeutung	Betätigung	Anzeige	Quittung	Anzeige nach Quittieren
1	Start Kommando 8	Cursor nach links, Hex.-Taste 8	8. ▲	↓	8.SA= XXXX ▲
2	Eingabe der Start-Adresse (SA)	4 Hex.-Tasten, z.B. 1,8,0,0	8.SA= 1800 ▲	↓	8.H 1= XXXX ▲
3	Eingabe der ersten Halt-Adresse (H1)	4 Hex.-Tasten, z.B. 1,8,3,C	8.H 1= 183C ▲	↓	8.H2= XXXX ▲
4	Eingabe der zweiten Halt-Adresse (H2)	4 Hex.-Tasten, z.B. 1,8, F,D	8.H2= 18Fd ▲	↓	8.PC= XXXX ▲ (siehe Text)

Bild 6.2.10: Kommando 8, Programm mit Unterbrechungs-Punkten (Halt-Adressen) starten.

Kommando 9: Programm im Einzelschritt abarbeiten (Bild 6.2.11)

Hier wird jeder Programmschritt einzeln durch Betätigen der Abschlußtaste ↓ gestartet, wenn der Cursor im Kommandofeld steht. Befindet sich der Cursor aber im Datenfeld, so bewirkt das Drücken einer der Abschlußtasten die Anzeige des nächsten Registerpaares (wie in Kommando 6). Dies ist in *Bild 6.2.11* als Schritt 2 und 3 gezeigt. Auf diese Weise können nach der Ausführung eines jeden Programmschrittes alle Register der CPU nacheinander betrachtet oder auch ihre Inhalte verändert werden. Zum Starten des nächsten Programmschrittes muß der Cursor ins Kommandofeld gebracht und dann die Abschlußtaste gedrückt werden; siehe Schritt 4 und 5 in *Bild 6.2.11;* hier ist als Beispiel die Anzeige des PSW dargestellt (Akkumulator und Flagbyte, AF). Zu Beginn kann der Program Counter PC z. B. im Kommando 6 mit der Start-Adresse des Programms geladen werden.

Auch hier können Anzeige und Tastatur nicht richtig benutzt werden.

Zuvor muß mit *Kommando 6* der PC mit der Start-Adresse (SA) geladen werden.

Schritt Nr.	Bedeutung	Betätigung	Anzeige	Quittung	Anzeige nach Quittieren (z. B:)
1	Start Kommando 9	Cursor nach links, Hex.-Taste 9	9. ▲	↓	9.SP = XXXX ▲
2	Cursor ins Datenfeld bringen	→	9.SP = XXXX ▲	↓	9.15 = XXXX ▲
3	Akku und Flag ins Datenfeld bringen	↓	9.15 = XXXX ▲	↓	9.AF = XXXX ▲
4	Cursor ins Kommandofeld	←	9.AF = XXXX ▲		
5	Programmschritt starten	↓	9.AF = YYYY ▲		

Bild 6.2.11: Kommando 9, Programm im Einzelschritt abarbeiten.

Kommandos A und B: Information auf bzw. von Magnetband übertragen (Bild 6.2.12)

Mit *Kommando A* wird der Speicherbereich zwischen der Anfangs-Adresse AA und der End-Adresse EA auf Magnetband (in der Regel auf einem Kassetten-Recorder) übertragen.

Mit *Kommando B* werden Informationen (Daten oder ein Programm) vom Band in den Speicherbereich zwischen AA und EA übertragen. Hier prüft das Monitor-Programm jedesmal das richtige Einschreiben (vgl. dazu Kommando 0).

Während des Übertragungs-Vorgangs zeigt das Datenfeld der ECB-85-Anzeige in schneller Folge die durchlaufenen Adressen an; dies ist in *Bild 6.2.12* durch die Zeichen * symbolisiert.

Es ist zu beachten: Beim *Kommando A* muß der Recorder *vor* der letzten Quittung (Schritt 3) gestartet werden; sonst gehen Informationen verloren.
Beim *Kommando B* wird der Recorder *nach* der letzten Quittung gestartet; denn das serielle Einlesen der Daten geschieht erst, nachdem das Monitor-Programm das Start-Bit erkannt hat.

Schritt Nr.	Bedeutung	Betätigung	Anzeige	Quittung	Anzeige nach Quittieren
1	Start Kommando A	Cursor nach links, Hex.-Taste A	A. ▲	↓	A.AA= XXXX ▲
2	Eingabe der Anfangs-Adresse (AA)	4 Hex.-Tasten, z.B. 1,8,0,0	A.AA= 1800 ▲	↓	A.EA= XXXX ▲
3	Eingabe der End-Adresse (EA)	4 Hex.-Tasten, z.B. 1,9,F,F	A.EA= 19FF ▲	↓	A.****= ** am Schluß: A. ▲ (siehe Text)

Bild 6.2.12: Kommando A, Information auf Band übertragen. Die Bedienung für Kommando B ist die gleiche.

Teil 2:
Übungen mit dem ECB 85

Vorbemerkung:

Kapitel 2 behandelte ausführlich die Darstellung von Informationen und Programmen in verschiedenen Codes. Insbesondere wurde dabei deutlich, daß alle über die ECB-85-Tastatur einzugebenden bzw. auf dem Anzeigefeld erscheinenden Informationen (Daten, Adressen) im Hex.-Code zu verstehen sind.
Dementsprechend sind in diesem Übungskapitel alle Zahlen, Daten und Adressen stets im Hex.-Code angegeben; wenn davon abweichend beispielsweise das Dezimalsystem benutzt wird, ist dies durch den Zusatz D hinter der Zahl symbolisiert; entsprechend bedeutet der Zusatz B die Angabe im Binär-Code.
Zur Eingabe von Daten und Programmen in den Speicher des ECB 85 sowie zum Starten von Programmen, zum Lesen der Registerinhalte, zum Übertragen von Speicherbereichen usw. wird im folgenden jeweils nur das dazu aufzurufende Monitor-Kommando genannt; die Einzelheiten des Kommando-Ablaufs sind in *Kapitel 6* ausführlich dargestellt worden.

Übung 1: Speicher-Kommandos

Hier werden einige Kommandos des ECB-85-Monitors durch praktische Arbeit mit dem Mikrocomputer geübt.

a)

Schreiben Sie mit *Kommando 1* in den Speicherbereich 1850 bis 18AA das Datenbyte AB.
Überprüfen Sie danach mit dem Kommando 0 den Inhalt folgender Adressen:

Adresse	Inhalt
1850	
188A	
18AB	

Es muß in den Speicherzellen von 1850 bis 18AA die Information AB stehen. (Ist dies nicht der Fall, dann wiederholen Sie bitte das Einschreiben mit *Kommando 1!*) In der Zelle 18AB dagegen kann eine beliebige Information stehen, weil Sie diese Zelle nicht beschrieben haben.

b)

Speichern Sie mit *Kommando 2* die Information aus dem RAM-Bereich 1850 bis 18AA in den RAM-Bereich ab Adresse 1A00 um!

Überprüfen Sie, ob das Kommando ausgeführt wurde, indem Sie mit *Kommando 0* die folgenden Speicherzellen lesen:

Adresse	Inhalt
1A00	
1A3A	
1A5B	

c)

Überspielen Sie mit *Kommando A* den RAM-Bereich 1A00–1A5A auf die Kassette.

d)

Lesen Sie nun zur Kontrolle mit *Kommando B* die Information vom Band in den Speicherbereich ab 1B31. (Berechnen Sie die Endadresse des Bereichs zur Übung selbst.) Überprüfen Sie anschließend mit *Kommando 0,* ob dies auch durchgeführt wurde. Notieren Sie hierzu die Information in:

Adresse	Inhalt
1B3F	
1B6A	
1B8C	

Lösung zur Übung 1:

Bei den Speicherzellen-Inhalten bedeutet X jeweils eine beliebige, zufällig im Speicher vorhandene Information, die mit den vorangegangenen Kommandos (oder Programmschritten) nicht beeinflußt wurde.

a)

1850:	AB
188A:	AB
18AB:	XX

b)

1A00:	AB
1A3A:	AB
1A5B:	XX

d)

Berechnung der Endadresse:

```
 1B31
+ 5A
 1B8B
```

Speicherzellen-Inhalte:

1B3F:	AB
1B6A:	AB
1B8C:	XX

Übung 2: Ein erstes Programm

Geben Sie folgendes kleine Programm mit *Kommando 0* in das RAM des ECB 85 ein (der NOP-Vorspann dient dem sicheren Überspielen auf Band; s. u.):

Speicheradresse	Hex.-Code	Mnemo-Code	
1800	00	NOP	
1801	00	NOP	
1802	00	NOP	
1803	00	NOP	
1804	00	NOP	
1805	00	NOP	
1806	00	NOP	
1807	00	NOP	
1808	00	NOP	
1809	00	NOP	
180A	00	NOP	
180B	0C	(Zahlenwert zwölf)	
180C	21	LXI	H,180B
180D	0B		
180E	18		
180F	46	MOV	B,M
1810	78	MOV	A,B
1811	87	ADD	A
1812	87	ADD	A
1813	80	ADD	B
1814	87	ADD	A
1815	2B	DCX	H
1816	77	MOV	M,A
1817	76	HLT	

Laden Sie nach der Eingabe des Programms den Programmzähler PC mit der Adresse 180C und bringen Sie danach den Akkumulator zur Anzeige *(Kommando 6)*.

Arbeiten Sie dann das Programm mit *Kommando 9* in Einzelschritten ab und beobachten Sie dabei den Akkumulator. Notieren Sie den Akkumulator-Inhalt immer, wenn er sich ändert.

A: A: A:
A: A: A:

Das Programm ist abgearbeitet, wenn im PC 1817 steht; kontrollieren Sie deshalb zwischendurch, ob Sie das Ende des Programms bereits erreicht haben (vgl. dazu die Beschreibung von *Kommando 9* in *Bild 6.2.11*).

Notieren Sie den Wert, der nach der Abarbeitung des Programms in der Adresse 180A steht:

Der Ausgangswert stand in der Zelle 180B. Wie lautet er?

Wenn der Wert in 180B der Ausgangswert ist und der Wert in der Zelle 180A das Ergebnis der arithmetischen Operationen darstellt, welche Aufgabe hat dann dieses Programm gelöst? Sie finden die Lösung vielleicht schneller, wenn Sie die oben notierten Zwischenwerte im Akkumulator betrachten und diese, um sie leichter zu übersehen, in die entsprechenden Dezimalzahlen umwandeln.

Retten Sie Ihr Programm, bevor Sie die Spannung abschalten, indem Sie es auf der Kassette speichern; überprüfen Sie, ob die Speicherung einwandfrei erfolgte, indem Sie das Programm zur Kontrolle vom Band in den RAM-Bereich 1A00 bis 1A17 laden und dort seine Richtigkeit überprüfen. Soll das Programm hier laufen, so ist jedoch der LXI-Befehl zu ändern, weil die Zahl zwölf jetzt in der Speicherzelle 1A0B steht. Es muß nun LXI H,1A0B H (im Hex.-Code: 21 0B 1A) heißen.

Lösung zur Übung 2:

		dezimal:	
A:	XX	YY	
A:	0C	12	
A:	18	24	(= 2 × 12)
A:	30	48	(= 4 × 12)
A:	3C	60	(= 5 × 12)
A:	78	120	(= 10 × 12)
180A:	78	120	
180B:	0C	12	

Der in der Zelle 180B stehende Wert zwölf wird durch mehrfache geschickte Addition mit zehn multipliziert. Dabei muß darauf geachtet werden, daß das Ergebnis der Multiplikation in 1 Byte paßt, also kleiner als 255D ist, so daß der Ausgangswert in 180B maximal gleich 25D sein darf (im Hex.-Code: 19).

Übung 3: Zwei kleine Arithmetik-Programme

a)
Auch in dieser Übung soll anhand von Register-Inhalten während eines Programmablaufs herausgefunden werden, welche Aufgabe das Programm erfüllt. Geben Sie dazu das folgende Programm ab Adresse 1850 ein.

Adresse	Hex.-Code	Mnemo-Code
1850	21	
1	70	LXI H,1870
2	18	
3	11	
4	70	LXI D,1870
5	18	
6	0E	MVI C,0A
7	0A	
8	46	MOV B,M
9	1A	LDAX D
A	90	SUB B
B	DA	
C	5F	JC 185F
D	18	
E	46	MOV B,M
F	13	INX D
60	23	INX H
1	0D	DCR C
2	C2	
3	59	JNZ 1859
4	18	
5	76	HLT

Dazu den Datenteil ab Adresse 1870:

1870	OD
1	1C
2	45
3	10
4	30
5	7F
6	8F
7	3F
8	F3
9	D2

Arbeiten Sie das Programm im Einzelschritt ab. Betrachten Sie während des Programmablaufs die Register B und C und notieren Sie jede Änderung. Das Programm ist abgearbeitet, wenn im Programmzähler die Adresse 1865 steht. Überlegen Sie jetzt, welche Funktion das Register C übernommen hat; vergleichen Sie dann die Inhalte des Registers B mit den Daten ab 1870; welche Beziehung besteht zwischen ihnen? Wenn Sie den Zusammenhang zwischen den Daten im Register B und im Datenteil nicht finden, dann ordnen Sie den Daten im Datenteil ihre Dezimaläquivalente zu und versuchen Sie es noch einmal.

Welche Aufgabe löst dieses Programm?

Lösung zur Übung 3a)
In den Registern B und C befinden sich beim Abarbeiten des Programms nacheinander folgende Werte:

B	C	B	C
XX	0A	7F	05
0D	0A	7F	04
0D	09	8F	04
1C	09	8F	03
1C	08	8F	02
45	08	F3	02
45	07	F3	01
45	06	F3	00
45	05		

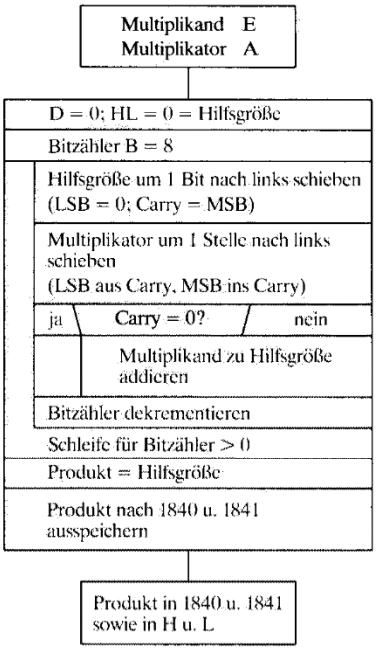
Wir erkennen, daß das Register C die Funktion eines Zählers erfüllt; in ihm wird von 0A bis 00 abwärts gezählt; bei Null ist der Programmablauf beendet.
Im Register B erscheinen zunächst die ersten drei Werte aus dem Datenteil; die Werte 10 und 30 werden dann ausgelassen, weil sie kleiner sind als der im Register B stehende Wert 45. Dagegen erscheinen dann 7F und 8F; sie sind größer als die vorhergehenden Werte. 3F ist wieder kleiner und wird ausgelassen, und zum Schluß steht F3, der größte Wert aus dem Datenteil, im Register B.
Es wird also mit dem Programm der größte Wert eines 10 Byte langen Datenfeldes gesucht und im Register B gespeichert.

Übung 3b)

Nun wollen wir noch ein kleines Multiplikationsprogramm durcharbeiten:

Adresse	Hex.-Code	Mnemo-Code
1820	16	MVI D,0
1	00	
2	21	LXI H,0
3	00	
4	00	
5	06	MVI B,8
6	08	
7	29	DAD H
8	17	RAL
9	D2	JNC 182D
A	2D	
B	18	
C	19	DAD D
D	05	DCR B
E	C2	JNZ 1827
F	27	
30	18	
1	22	SHLD 1840
2	40	
3	18	
4	76	HLT

Notieren Sie, welche Information in den Speicherzellen 1840 und 1841 steht.
Laden Sie nun den Akkumulator mit der Dezimalzahl 22, das Register E mit der Dezimalzahl 11, und lassen Sie das Programm in Echtzeit (d. h. im Kommando 7) ablaufen. Das Programm läuft auf den Haltbefehl, und der Prozessor kann durch Betätigen der RESET-Taste aus dem Haltzustand gebracht werden. Drücken Sie deshalb RESET.
Sie finden dann mit Kommando 0 in 1840: F2 und in 1841: 00, d. h. das Ergebnis der Multiplikation ist 00F2H = 242D, also 22D × 11D.
Das Programm multipliziert den Inhalt des Akkumulators mit dem Inhalt des Registers E. Da das Ergebnis der Multiplikation 2 Byte lang sein kann, wird das zunächst in H und L stehende Ergebnis unter 1840 und 1841 abgelegt, und zwar so, daß unter 1841 das höherwertige Ergebnisbyte und unter 1840 das niederwertige steht. Überprüfen Sie nun, daß auch für andere Werte in A und E die Multiplikation richtig durchgeführt wird!
Aus dem Struktogramm ist der genaue Ablauf des oben aufgelisteten Programms zu erkennen.



Struktogramm zu *Übung 3b);* LSB = Least Significant Bit, MSB = Most Significant Bit.

Übung 3c)

Um dieses Programm auch später bei Bedarf verfügbar zu haben, sollte man es in einem EPROM speichern.
Legen Sie dazu ein EPROM vom Typ 2716 in den grünen Sockel und wählen Sie das *Kommando 2.* Nach Betätigung der Quittungstaste fordert der Computer die Anfangsadresse; geben Sie 1820 ein und quittieren Sie (vgl. *Bild 6.2.4*).
Die Endadresse des Programms ist 1834, geben Sie also für EA 1834 ein; als Bestimmungsadresse wählen Sie eine Adresse im freien Bereich Ihres EPROMs, dessen Adreßbereich bei 0800 bis 0FFF liegt, z. B. bA = 0820. Legen Sie dann den Schalter S1 auf Programmieren und quittieren Sie (↓). Sie beobachten jetzt, wie das Programm in das EPROM geschrieben wird (schnell durchlaufende Anzeige im Adreß- und im Datenfeld). Wenn nur noch die 2 im Kommandofeld blinkt, dann ist alles einwandfrei gelaufen, und Sie können den Schalter S1 zurückstellen.
Bei späterer Benutzung muß das Programm wegen der Sprungadressen wieder in den alten Speicherbereich (1820–1834) übertragen werden.

Übung 4: Der Ablauf arithmetischer Befehle mit Flagbyte-Auswertung

a)

Es werden (mit *Kommando 0*) folgende Befehlsworte eingegeben:

Adresse	Hex.-Code	Mnemo-Code
1900	3E 14	MVI A,20D
1902	C6 42	ADI 66D
1904	C6 BE	ADI –66D

Dabei sind die Dezimalzahlen 66 und –66 wie folgt umzurechnen:
66 D = 01000010 B = 42 H
–66 D = 10111101 B + 1 B = 10111110 B = BE H

Mit *Kommando 6* ist dann der Programmzähler mit dem Wert 1900 zu laden und dann AF (Akku + Flagbyte) in die Anzeige zu bringen.

Nun wird das Programm mit *Kommando 9* im Einzelschritt abgearbeitet und dabei Akku und Flag betrachtet. Es ergeben sich nacheinander folgende Werte:

AF = 14 XX	Im 1. Progr.-Schritt wurde der Akku mit 14 H (= 20 D) geladen.
AF = 56 04	Zum Akku-Inhalt wurde 42 H (= 66 D) addiert; das Parity-Bit wurde gesetzt, alle anderen Flagbits zurückgesetzt.
AF = 14 15	Es wurde BE (= –66 D) addiert, so daß das Ergebnis gleich dem Ausgangswert ist (14 H = 20 D).

Nach dem letzten Programmschritt hat das Flagbyte folgenden Inhalt (zur Benennung vgl. *Bild 4.2.1*):

	S	Z	–	AC	–	P	–	C
	0	0	X	1	X	1	X	1
Hex.-Code:		1				5		

Dabei können die mit X bezeichneten Bits 0 oder 1 sein, so daß statt 15 auch 3F in der Anzeige stehen könnte (wenn nämlich alle X-Bits 1 wären).

b)

Es werden (mit *Kommando 0*) folgende Befehlsworte eingegeben:

Adresse	Hex.-Code	Mnemo-Code
1A00	3E 56	MVI A,56H
1A02	CE BE	ACI –66D
1A04	CE 42	ACI 66D

Dabei sind die Dezimalzahlen 66 und –66 wie bei *a)* umzurechnen, und der Start und Ablauf des Programms geschieht ebenfalls wie oben beschrieben. Akku und Flag lauten nach dem ersten Programmschritt:

AF = 56 XX	Der MVI-Befehl verändert die Flag-Bits (Bedingungs-Bits) nicht.

Nun sind zwei Fälle zu unterscheiden: Das Carry-Bit war vorher gesetzt oder nicht.

Fall 1: Carry-Bit war vorher *nicht* gesetzt:

AF = 14 15	Zum Akku-Inhalt wurde BE H (= –66 D) addiert.
AF = 57 00	Addition von 42 H (= 66 D); das Ergebnis ist um 1 größer als der Ausgangswert, da bei dieser Addition das nach Schritt 2 (ACI –66 D) gesetzte Carry-Bit ebenfalls addiert wird.

Fall 2: Carry-Bit war vorher *gesetzt:*

AF = 15 11	Zum Akku-Inhalt wurde BE H (= –66 D) *und* das Carry-Bit addiert.
AF = 58 00	Das Ergebnis ist um 2 größer als der Ausgangswert (56 H), da in *beiden* Schritten das Carry-Bit ebenfalls zum Akku-Inhalt addiert wurde.

c)

Es werden (mit *Kommando 0*) folgende Befehlsworte eingegeben:

Adresse	Hex.-Code	Mnemo-Code
1B00	3E 00	MVI A,0
1B02	D6 01	SUI 1

Der Start und Ablauf des Programms geschieht hier ebenfalls wie oben beschrieben.
Akku und Flag lauten:

AF = 00 XX	Der Akku wurde mit 0 geladen (MVI A,0).
AF = FF 85	Es wurde 1 subtrahiert (SUI 1).

Anhand der Binärdarstellung der Zahlen wird die Operation nachstehend erläutert:
Zahlenwert eins: 00000001 B
Für den Zahlenwert –1 ist das Zweier-Komplement zu bilden (bitweise invertieren und dann 1 addieren):
1 D = 00000001 B
–1 D = 11111110 B + 1 B = 11111111 B
Im Prozessor läuft folgendes ab:

Akku mit 0 laden:	00000000 B
–1 addieren:	+ 11111111 B (= –1 D)
Resultat:	11111111 B = –1 D = FF H

Das Flag-Byte hat nach Ablauf des Programms folgenden Inhalt:

S	Z	–	AC	–	P	–	C
1	0	X	0	X	1	X	1
		8			5		

Hex.-Code:

Hierbei ist angenommen, daß alle X-Bits 0 sind.

d)

Es werden (mit *Kommando 0*) folgende Befehlsworte eingegeben:

Adresse	Hex.-Code	Mnemo-Code
1850	3E 00	MVI A,0
1850	DE 01	SBI 1

Der Start und Ablauf des Programms geschieht hier ebenfalls wie oben beschrieben.
Akku und Flag lauten nach dem ersten Programmschritt:

AF = 00 XX	Der Akku wurde mit 0 geladen.

Nun sind zwei Fälle zu unterscheiden: Das Carry-Bit war vorher gesetzt oder nicht.

Fall 1: Carry-Bit war vorher *nicht* gesetzt:

AF = FF 85	Es wurde 1 vom Akku-Inhalt subtrahiert.

Fall 2: Carry-Bit war vorher *gesetzt:*

AF = FE 81	Es wurde 1 und das Carry-Bit vom Akku-Inhalt subtrahiert.

Für den Fall 1 gelten die unter *c)* gemachten Ausführungen, d. h. die Subtraktion von 1 ergibt im Akku: FF H = 11111111 B = –1 D.
Bei *Fall 2* läuft im Prozessor folgendes ab:

Akku mit 0 laden:	00000000 B
–1 addieren:	+ 11111111 B (= –1 D)
Carry-Bit subtr.	– 1 B
Resultat:	11111110 B = –2 D = FE H

Das Flag-Byte bei *Fall 2* hat nach Ablauf des Programms folgenden Inhalt:

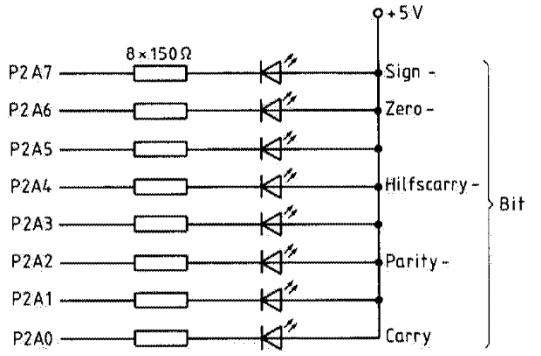
S	Z	–	AC	–	P	–	C
1	0	X	0	X	0	X	1
		8			1		

Hex.-Code:

Hierbei ist angenommen, daß alle X-Bits 0 sind.

e) Zusatz für Fortgeschrittene

Es ist natürlich auch möglich, die in *a)* bis *d)* beschriebenen arithmetischen Befehle im Programm ausführen zu lassen und dabei das Flag-Byte anzuzeigen. Dieses wird dazu beispielsweise auf den Port A des 8155 Nr. 2 ausgegeben (und zwar invertiert, s. u.) und mit Hilfe von 8 Leuchtdioden angezeigt. Es empfiehlt sich, das Programm dann in *Kommando 8,* also mit Haltepunkten, ablaufen zu lassen.
Zunächst ist die in der Zeichnung dargestellte Schaltung aufzubauen; hier sind die Port-Anschlüsse angegeben (vgl. *Bild 3.4.1*).



Schaltung zur Anzeige der Flag-Bits

Zum Programm:

Das Unterprogramm UNT sei im Anwender-EPROM ab Adresse 084F abgelegt und wird daher mit CD 4F 08 (Hex.-Code) vom Hauptprogramm aufgerufen. Vor der Ausgabe des Akku-Inhalts ist dieser mit CMA zu invertieren, da in der oben gezeigten Schaltung jede Diode dann leuchtet, wenn der mit ihr verbundene Port-Anschluß 0-Wert (L-Pegel) führt; so entspricht einer leuchtenden Diode ein 1-Wert des betr. Flag-Bits (0-Wert nach Inversion).
Das Hauptprogramm ist für die Teile *a)* und *b)* dieser *Übung 4* weiter unten aufgelistet. Die Haltepunkte beim Ablauf des Kommandos 8 sind die Adressen, die auf den CALL-Befehl folgen, also 190A, 190F, 1914 und 1919. Es wird nach Ausführung jedes arithmetischen Befehls das Programm angehalten, und das Bitmuster des Flag-Bytes kann in Ruhe betrachtet werden.

Ablauf der Übung:

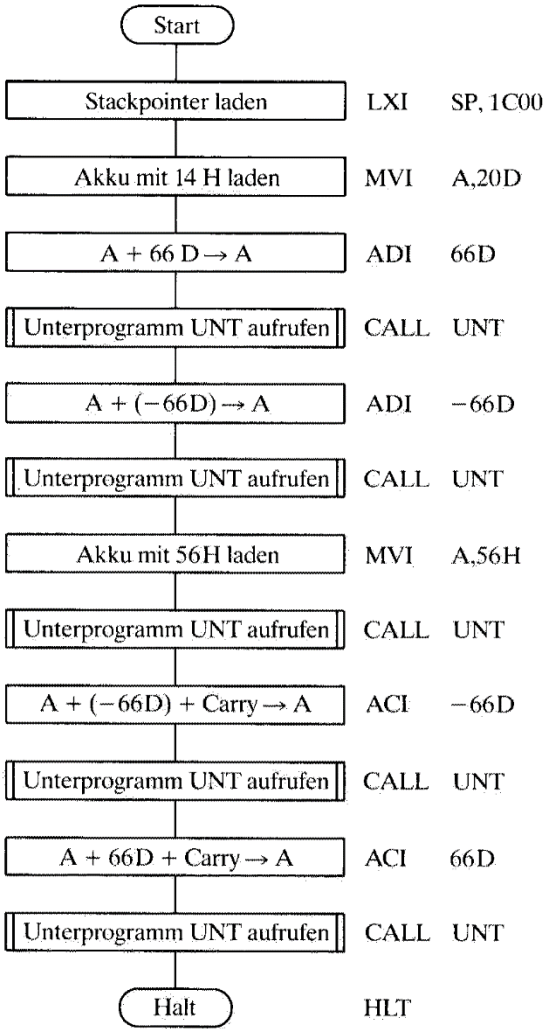
1. Eingabe des Hauptprogramms, z. B. ab Adresse 1900;
2. Abspeichern des Unterprogramms im EPROM (ab 084F);
3. Start des Programms in *Kommando 8* (Start-Adresse 1900, Haltepunkte wie oben beschrieben).

Hauptprogramm:

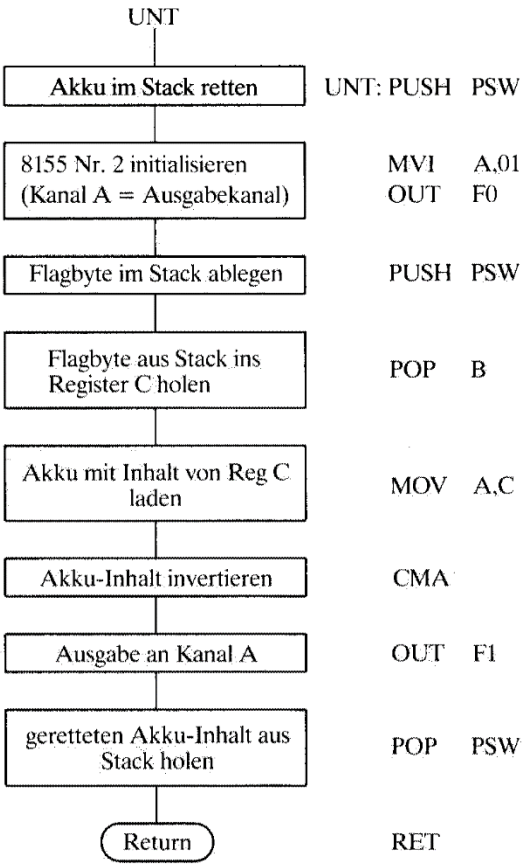
Adr.	Hex.-Code	Mnemo-Code
1900	31 001C	LXI SP,1C00
1903	3E 14	MVI A, 20D
1905	C6 42	ADI 66D
1907	CD 4F08	CALL UNT
190A	C6 BE	ADI −66D
190C	CD 4F08	CALL UNT
190F	3E 56	MVI A,56
1911	CD 4F08	CALL UNT
1914	CE BE	ACI −66D
1916	CD 4F08	CALL UNT
1919	CE 42	ACI 66D
191B	CD 4F08	CALL UNT
191E	76	HLT

Unterprogramm:

Adr.	Hex.-Code	Label	Mnemo-Code
084F	F5	UNT:	PUSH PSW
0850	3E 01		MVI A,01
0852	D3 F0		OUT F0
0854	F5		PUSH PSW
0855	C1		POP B
0856	79		MOV A,C
0857	2F		CMA
0858	D3 F1		OUT F1
085A	F1		POP PSW
085B	C9		RET



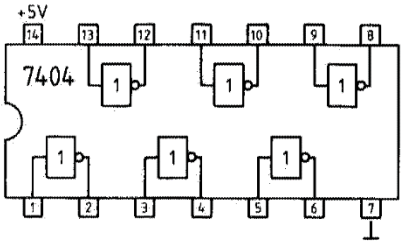
Flußdiagramm und Mnemo-Code des Hauptprogramms.



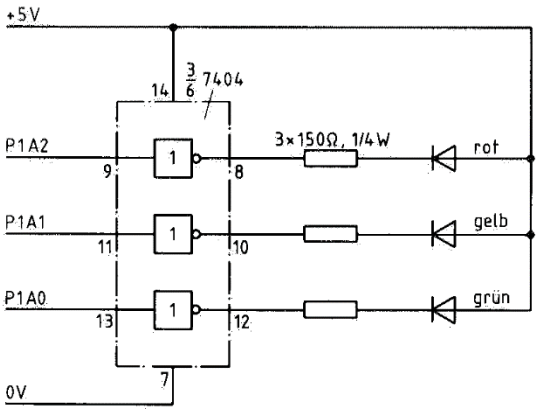
Flußdiagramm und Mnemo-Code des Unterprogramms.

Übung 5: Ansteuerung von drei Leuchtdioden und Auswertung von drei Schalter-Signalen

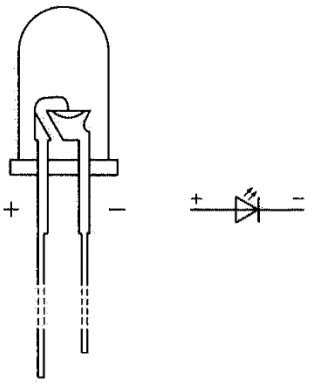
Als Vorbereitung zur *Übung 6* sollen hier drei Leuchtdioden über den Port A des 8155 Nr. 1 angesteuert werden, und es sind die Zustände von drei Schaltern über den Port C desselben E/A-Bausteins auszuwerten. In den Abbildungen sind die aufzubauenden Schaltungen sowie der Anschlußplan des 7404-ICs (6 NICHT-Glieder) wiedergegeben, so daß der Aufbau, z. B. auf einer Lochraster-Platine, keine weiteren Schwierigkeiten bieten sollte. Aus den Zeichnungen geht des weiteren hervor, daß die drei Leuchtdioden an die drei niedrigstwertigen Bits des Ports A (8155 Nr. 1) angeschlossen werden und die Schalter an die drei niedrigstwertigen Bits des Ports C. Die Belegung der Anwender-Schnittstelle geht aus *Bild 3.4.1* hervor, so daß es hier ausreicht, die Port-Anschlüsse anzugeben, z. B. P1A0, P1A1, usw.



Anschlußplan des ICs 7404, Pin-Belegung von oben gesehen.



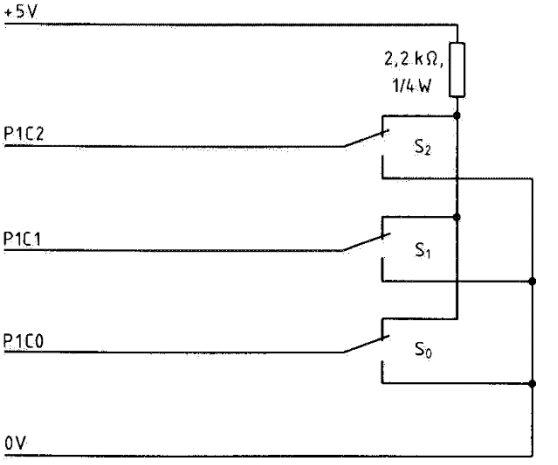
Verschaltung der drei Leuchtdioden an Port A des 8155 Nr. 1.



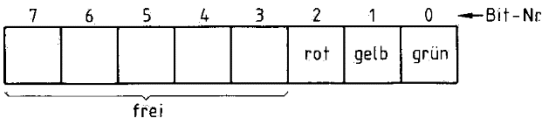
Aufbau der Leuchtdioden. Die Anode (+) ist der längere Anschluß; auf richtige Polung ist unbedingt zu achten. Rechts das Schaltzeichen der Leuchtdioden.

Funktionstest für die Dioden-Schaltung:

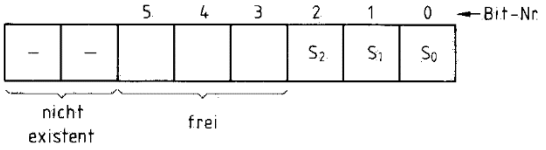
Zunächst ist der 8155 Nr. 1 zu initialisieren; vgl. hierzu *Bild 3.3.6.* Weil Port A zur Ausgabe und Port C zur Eingabe benutzt werden soll, lautet das Steuerwort binär XXXX0001, bzw. hexadezimal 01, wenn alle hier bedeutungslosen X-Bits Null gesetzt werden. Dieses Steuerwort 01 ist also z. B. mit *Kommando 4* (siehe *Bild 6.2.6*) in das Steuerwort-Regi-



Verschaltung der drei Schalter an Port C des 8155 Nr. 1.



Belegung von Port A mit den drei Leuchtdioden.



Belegung von Port C mit den drei Schaltern. Port C hat nur 6 Bits.

ster des 8155 Nr. 1 mit der Port-Adresse F8 einzugeben. Danach wird dann das gewünschte Datenbyte an den Port A mit der Adresse F9 ausgegeben. Hiermit können nun die drei Leuchtdioden angesteuert werden; die NICHT-Glieder im 7404-IC bewirken, daß die Kathode der betr. LED mit L-Pegel verbunden wird (die Diode also leuchtet), wenn das betr. Port-Bit 1-Wert führt.

Ausgabe mit Kommando 4 (vgl. Bild 6.2.6):

F9 = 01: grüne Diode muß leuchten
F9 = 02: gelbe Diode muß leuchten
F9 = 04: rote Diode muß leuchten.
Ist dies nicht der Fall, so liegt in der Schaltung ein Fehler vor; z. B. können die Dioden verpolt oder das IC ohne Stromversorgung sein. Wenn die Schaltung einwandfrei arbeitet, versuchen Sie nun, die rote und die gelbe Diode gleichzeitig zum Leuchten zu bringen. Dazu ist, wie Sie sehen werden, der Wert 06 an Port A auszugeben. Machen Sie sich dazu klar, daß die 8 Bits des Ports A als 8stellige Dualzahl aufzufassen sind, wobei die Bit-Nr. (siehe Abbildung) die Zweierpotenz im Dualsystem angibt: Liegen die Bits 2 und 1 beide auf 1, so ergibt sich als Zahlenwert: $1 \times 2^2 + 1 \times 2^1 = 6$.
Entsprechend müssen bei der Ausgabe F9 = 05 die rote und die grüne Diode leuchten.

Funktionstest für die Schalter:

Mit der beim Funktionstest der Leuchtdioden-Schaltung beschriebenen Initialisierung des 8155 Nr. 1 wurde schon festgelegt, daß Port C zur Eingabe benutzt werden soll. So ist es nun ohne weiteres möglich, mit *Kommando 5* den Zustand von Port C abzufragen bzw. in die Anzeige des ECB 85 zu bringen.
In binärer Schreibweise lautet das Byte dann: XXXX XYYY, wobei XXXX X für den vorliegenden Fall bedeutungslose Bits sind (das höherwertige Halbbyte wird im folgenden in der hexadezimalen Schreibweise ebenfalls mit X bezeichnet); YYY sind die drei Bits, die den Schalterstellungen entsprechen.

Einlesen mit Kommando 5 (vgl. Bild 6.2.7):

Die Port-Adresse des Ports C vom 8155 Nr. 1 ist FB (siehe *Bild 3.3.5*), so daß im *Kommando 5* wie folgt eingelesen wird (nach jeder Änderung der Schalterstellungen ist die Quittungstaste zu drücken; es ist hier P1C3 = 0 vorausgesetzt):
FB = X1: $S_2 = 0, S_1 = 0, S_0 = 1$
FB = X2: $S_2 = 0, S_1 = 1, S_0 = 0$
FB = X4: $S_2 = 1, S_1 = 0, S_0 = 0$
FB = X5: $S_2 = 1, S_1 = 0, S_0 = 1$
Auch hier gilt das beim Port A über den Zusammenhang mit den Dualzahlen Gesagte.
Zu beachten ist, daß das Betätigen der RESET-Taste die Initialisierung der 8155-Bausteine aufhebt, das Steuerwort-Register danach also erst wieder neu beschrieben werden muß (s. o.).

Funktionsprüfung der Schalter mit Hilfe eines kleinen Programms:

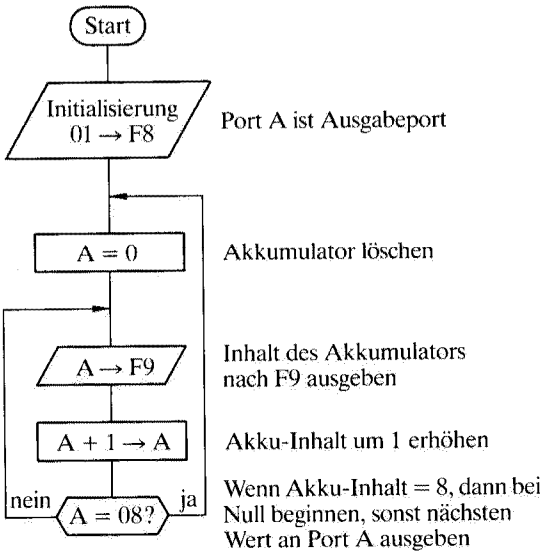
Adresse	Hex.-Code	Mnemo-Code	Bemerkung
1900	3E	MVI A,01	Steuerwort
1	01		in Akku
2	D3	OUT F8	Steuerwort
3	F8		ausgeben
4	DB	IN FB	Schaltersignale
5	FB		einholen
6	D3	OUT F9	und an Dioden
7	F9		ausgeben
8	C3	JMP 1904	Sprung zum
9	04		Einlesen
A	19		

Schalterstellungen werden an Leuchtdioden ausgegeben.

Das vorstehend aufgelistete Programm liest die Schalterstellungen vom Port C des 8155 Nr. 1 ein und gibt die eingelesenen Bits an Port A aus. Da hier eine Endlos-Schleife vorliegt (Rücksprung zum Einlese-Befehl, Adresse 1904), kann das Programm nur mit der RESET-Taste abgebrochen werden, nachdem es mit *Kommando 7* gestartet wurde.

Programmierung einer Dualzähler-Anzeige:

Nach dem weiter oben über den Zusammenhang zwischen den Port-Bits und der Wertigkeit im Dualsystem Gesagten ist klar, daß man der an P1A0 angeschlossenen Leuchtdiode den Wert 1, der an P1A1 den Wert 2 und der Diode an P1A2 den Wert 4 zuordnen kann. Mit den drei Leuchtdioden kann also ein Dualzähler mit drei Stellen realisiert werden, der von 000 (null) bis 111 (sieben) läuft. Im unten wiedergegebenen Programm wird dazu der Inhalt des Akkumulators innerhalb einer Schleife jeweils um 1 erhöht und dann an den Port A des 8155 Nr. 1 ausgegeben. Die Initialisierung des E/A-Bausteins kann hier natürlich auch im Programm erfolgen (mit MVI A,01 und OUT 08, wie im vorangegangenen Programmbeispiel auch). Noch eine Bemerkung zum Befehl XRA A: dieser löscht den Akkumulator-Inhalt, weil bei der Exklusiv-ODER-Verknüpfung gleiche Bits stets 0 ergeben, und hier wird ja der Akku-Inhalt mit sich selbst EXOR-verknüpft. Arbeiten Sie das Dualzähler-Programm im *Einzelschritt-Kommando 9* ab und notieren Sie, was Sie sehen.



Flußdiagramm zur Dualzähler-Anzeige.

Adresse	Hex.-Code	Mnemo-Code
1811	3E	MVI A,01
2	01	
3	D3	OUT F8
4	F8	
5	AF	XRA A
6	D3	OUT F9
7	F9	
8	3C	INR A
9	FE	CPI 08
A	08	
B	CA	JZ 1815
C	15	
D	18	
E	C3	JMP 1816
F	16	
20	18	

Programm zur Dualzähler-Anzeige

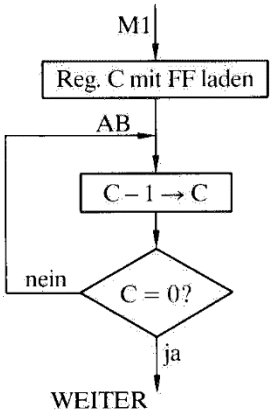
Übung 6: Steuerung einer Verkehrsampel

In der vorangegangenen Übung wurden drei Leuchtdioden als Dualzähler-Anzeige benutzt, und das Programm wurde im Einzelschritt abgearbeitet.
Nun soll ein Programm erstellt werden, das die drei Leuchtdioden wie eine *Verkehrsampel* arbeiten läßt. Weil hier der Einzelschrittbetrieb aus naheliegenden Gründen nicht sinnvoll ist, muß das Programm auch den Ablauf von (unterschiedlich langen) Zeitspannen bzw. Phasen realisieren:
Rot-Phase
Rot/Gelb-Phase
Grün-Phase
Gelb-Phase.
(Anschließend erfolgt Rücksprung zur Rot-Phase).

Man braucht also einen Programmteil, mit dem der Ablauf einer Zeitspanne realisiert wird. Jeder Befehl in einem Programm benötigt für seine Ausführung eine bestimmte Anzahl von Taktzyklen (siehe *Kapitel 3.1.2*), die in der Befehlsliste ab *Seite 140* jeweils angegeben ist. Um Zeit zu verbrauchen, kann man etwa einen Befehl abarbeiten lassen, der keinen Datentransport, keine logische oder arithmetische Operation und auch sonst nichts weiter bewirkt, z. B. den Befehl MOV A,A; er braucht 4 Taktzyklen, d. h. 4×325 ns = 1,3 μs. Für irgendwelche praktischen Zwecke wie die hier gewünschte Ampelphase ist eine so kurze Zeitspanne von 1,3 Mikrosekunden viel zu kurz, und es müßten für lediglich eine einzige Millisekunde schon 770 solche Befehle hintereinander ausgeführt werden.
Natürlich wäre es sowohl vom Speicherplatzbedarf als auch von der Programmierung her gesehen unwirtschaftlich und unsinnig, Tausende gleichlautender, eigentlich bedeutungsloser Befehle hintereinander in ein Programm zu schreiben. Man geht daher zum Realisieren von Zeitspannen im Programm folgenden Weg: ein Register der CPU (z. B. das Register C) wird mit einem Anfangswert geladen; dann wird der Registerinhalt um eins erniedrigt und darauf geprüft, ob er schon null ist; wenn nicht, wird wieder um eins erniedrigt, usw. – bis der Wert null erreicht ist. Erst dann wird diese sogen. Schleife verlassen. *Mit anderen Worten:* Man läßt den Computer von einem Anfangswert bis auf null herunterzählen und kann so sehr einfach bestimmen, wie lang die Zeitspanne wird. Wie dies in einem Programm zu realisieren ist, wird hier im Flußdiagramm und im Programm (Mnemo-Code) gezeigt. Dabei ist für jeden Befehl angegeben, wieviele Taktzyklen seine Ausführung benötigt. Beim bedingten Sprungbefehl JNZ AB sind das 10 Takte, wenn die Bedingung erfüllt ist (der Sprung also ausgeführt wird), und 7 Takte im anderen Fall.
Der gesamte Zeitbedarf für die in der Abbildung dargestellte Zeitschleife wird wie folgt berechnet:

1 *	7 Takte	für MVI C,FF	(Anfangswert 255 D laden)
1 *	4 Takte	für DCR C	(erste Ausführung)
254 *	4 Takte	für DCR C	(bis Null erreicht ist)
254 *	10 Takte	für JNZ AB	(Bed. erfüllt, Rücksprung)
1 *	7 Takte	für JNZ AB	(Bed. nicht erfüllt)

3574 Takte für die gesamte Zeitschleife; das sind 3754 * 325 Nanosekunden = 1,162 Millisekunden.



Label	Mnemo-Code	Takte
M 1:	MVI C,FF	7
AB:	DCR C	4
	JNZ AB	10/7
WEITER:		

Flußdiagramm und Programm für eine Zeitschleife (1,162 ms).

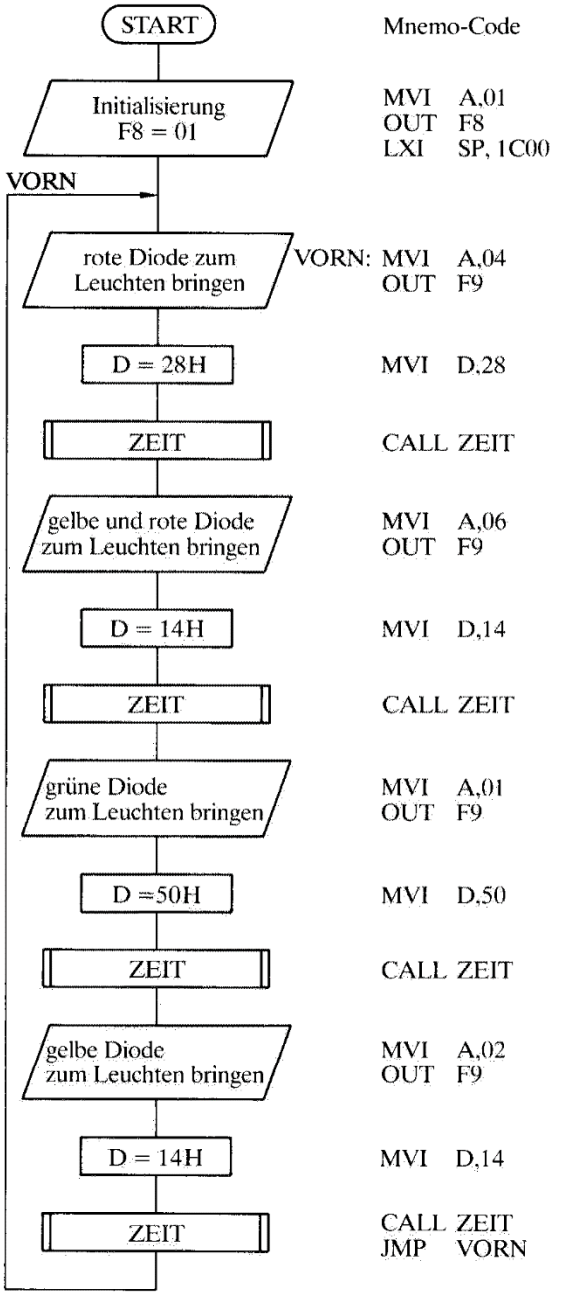
Aus dem Zeitbedarf von 325 ns pro Taktzyklus kann man leicht errechnen, daß für eine Millisekunde 3077 Taktzyklen benötigt werden. Für diesen Fall soll nun berechnet werden, wie der Anfangswert im Register C für die obige Zeitschleife zu wählen ist.
Im gegebenen Beispiel (Anfangswert 255 D = FF H) ist die Anzahl der Takte:
3574 = 1 * (7 + 4 + 7) + 254 * (4 + 10) bzw.:
3574 = 18 + 254 * 14
Für eine Millisekunde ergibt sich entsprechend:
3077 = 18 + X * 14,
wobei X der Anfangswert im Register C minus 1 ist (254 D im obigen Beispiel).
Aus dieser Gleichung ergibt sich X = 218,5. Das bedeutet, daß man das Register C mit dem Startwert X + 1, also 219 D = DB H oder 220 D = DC H laden muß, um mit einem Fehler von ca. 0,3 % eine Zeitspanne von einer Millisekunde zu realisieren.
Die Zeitschleife für ca. eine Millisekunde lautet also:

```
MS:     MVI C,DB
AB:     DCR C
        JNZ AB
```

Läßt man diese Zeitschleife nun entsprechend oft ablaufen, d. h. legt man um sie herum eine weitere Schleife, so kann man eine Zeitspanne von maximal 256 ms realisieren (da ein CPU-Register ein Byte enthält, also mit höchstens 255 D = FF H geladen werden kann). Es bietet sich hier an, die Millisekunden-Zeitschleife 250mal ablaufen zu lassen, so daß eine Viertelsekunde resultiert. Dazu ist z. B. das Register B mit 249 D = F9 H zu laden und ebenfalls bis Null herunterzuzählen. Die Viertelsekunden-Schleife sieht dann so aus:

```
        MVI B,F9
MS:     MVI C,DB
AB:     DCR C
        JNZ AB
        DCR B
        JNZ MS
```

Im Ampelprogramm werden, wie schon erwähnt, unterschiedlich lange Zeitspannen benötigt. Dazu ist es zweckmäßig, die Viertelsekunden-Schleife als *Unterprogramm* anzulegen, in dem ein weiteres CPU-Register heruntergezählt wird, das aber im aufrufenden Hauptprogramm geladen wird. Damit können Vielfache von Viertelsekunden realisiert werden (von 1 bis zu 255 Viertelsekunden). Wird im Hauptprogramm das Register D geladen, dessen Startwert die Vielfachen der Viertelsekunden angibt, dann lautet das mit dem Befehl CALL ZEIT



Flußdiagramm und Programm für den Betrieb einer Verkehrsampel. Zur Hardware-Verschaltung siehe *Übung 5*.

vom Hauptprogramm aufzurufende Zeitschleifen-Unterprogramm:

```
ZEIT:   MVI B,F9
MS:     MVI C,DB
AB:     DCR C
        JNZ AB
        DCR B
        JNZ MS
        DCR D
        JNZ ZEIT
        RET
```

Hiermit wird also eine Zeitspanne realisiert, die so viele Viertelsekunden dauert, wie der im Hauptprogramm in das Register D geladene Anfangswert angibt.
Mit den in der vorangegangenen Übung gegebenen Informationen und dem eben erarbeiteten Unterprogramm ZEIT kann nun das Ampelprogramm erstellt werden. Die vorstehende Abbildung zeigt das Flußdiagramm und das Programm im Mnemo-Code; übertragen Sie es in den Hex.-Code, geben Sie es in den ECB 85 ein und testen Sie es.

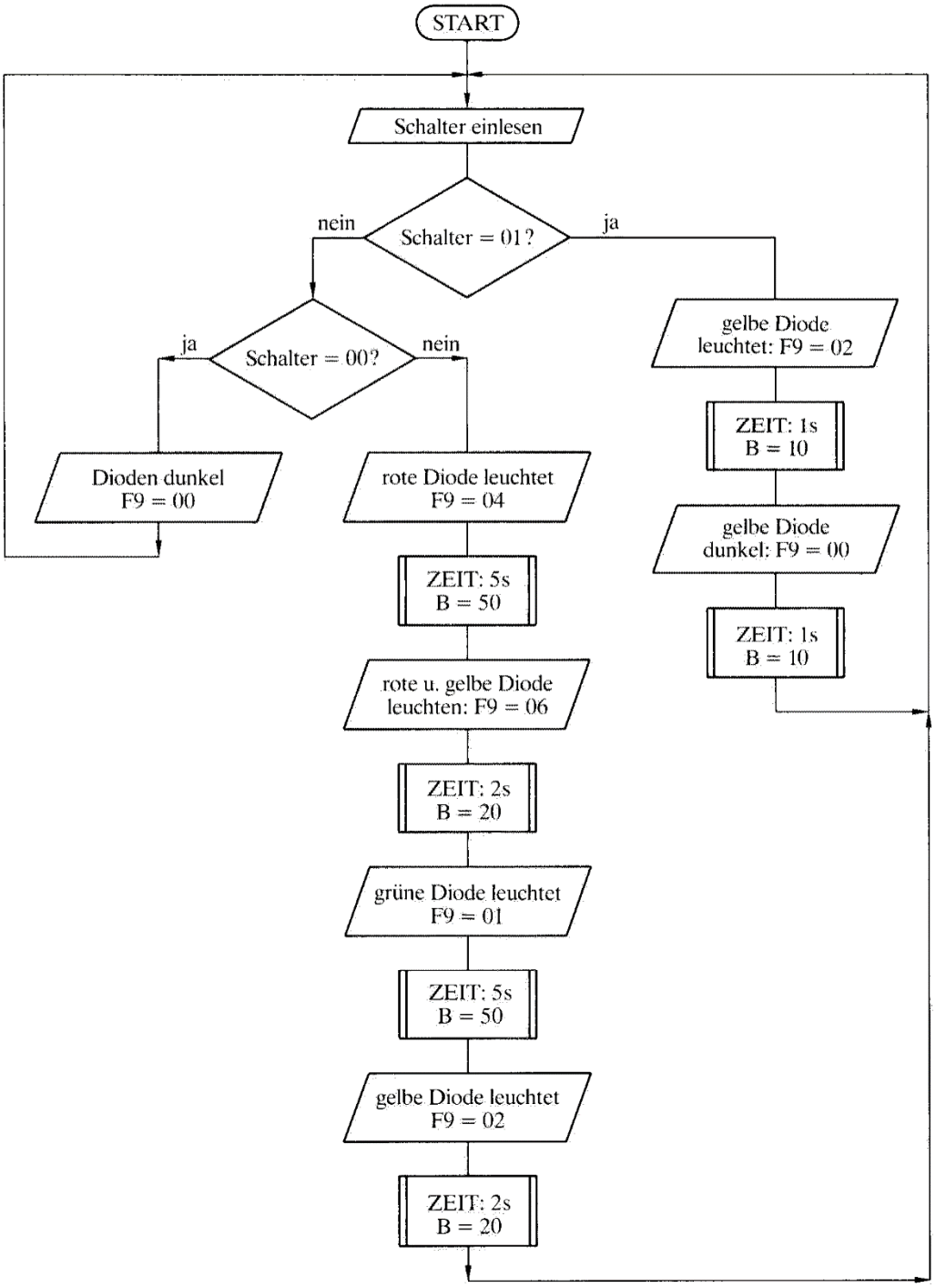
Nun ist das Ampelprogramm zu erweitern, und zwar soll zwischen *Tagbetrieb* (wie eben beschrieben), *Nachtbetrieb* (gelbes Blinklicht) und *Abschalten* zu wählen sein. Dazu dienen drei Eingabeschalter an Port C, wie sie in *Übung 5* benutzt wurden. Der von Port C eingelesene Zahlenwert (der der Schalterstellung entspricht) soll folgende Bedeutung haben:
0: aus
1: gelbes Blinklicht
2 oder 3: normaler Ampelbetrieb
Für das Programm, mit dem dies realisiert wird, ist nachfolgend das Flußdiagramm und das komplette Listing wiedergegeben. Dazu noch eine Anmerkung: Um die Vielfalt der Programmierungsmöglichkeiten zu zeigen, ist mit dem Unterprogramm ZEIT hier eine Zeitspanne von einer Zehntelsekunde realisiert (im Unterschied zur oben vorgestellten Variante), und der ins Register B geladene Wert gibt an, wieviele Zehntelsekunden ablaufen sollen; weiterhin werden Programmbefehle und damit Speicherplätze gespart, indem der OUT-Befehl, der die Leuchtdioden ansteuert, im Unterprogramm enthalten ist.

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
1950	D3 F9	ZEIT:	OUT F9	
52	0E 64	AA:	MVI C,64	
54	16 DB	AB:	MVI D,DB	} Zeitschleife 1 ms
56	15	AC:	DCR D	
57	C2 56 19		JNZ AC	
5A	0D		DCR C	
5B	C2 54 19		JNZ AB	
	05		DCR B	
5F	C2 52 19		JNZ AA	
62	C9		RET	

Unterprogramm ZEIT zur Ampelsteuerung

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
18E0	3E 01		MVI A,01	} Port A: Ausgabe
E2	D3 F8		OUT F8	} Port C: Eingabe
E4	31 00 1C		LXI SP, 1C00	Stackpointer adressieren
E7	DB FB	ABF:	IN FB	Schalterstellung einholen
E9	E6 03		ANI 03	Bitpos. 3–7 auf 0 setzen
EB	FE 01		CPI 01	Abfrage der Schalter (Subtraktion von 1)
ED	CA 20 19		JZ GELB	bei 0: Schalter = 1 (≙ Blinken)
F0	F2 00 19		JP AMPEL	bei > 0: Schalter > 1 (≙ Ampel)
F3	3E 00	NULL:	MVI A,00	} (< 0: aus):
F5	D3 F9		OUT F9	} Ausgabe von Null: Dioden aus
F7	C3 E7 18		JMP ABF	Sprung zur Abfrage
FA	00 00 00		NOP	
FD	00 00 00		NOP	
1900	3E 04	AMPEL:	MVI A,04	Laden des Akkus mit 04 (≙ rot)
02	06 32		MVI B,32	} Laden von B mit 50 D,
04	CD 50 19		CALL ZEIT	} 5 Sekunden
07	C6 02		ADI 02	Laden des Akkus mit 06 (≙ rot/gelb)
09	06 14		MVI B,14	} Laden von B mit 20 D,
0B	CD 50 19		CALL ZEIT	} 2 Sekunden
0E	D6 05		SUI 05	Laden des Akkus mit 01 (≙ grün)
10	06 32		MVI B,32	} Laden von B mit 50 D,
12	CD 50 19		CALL ZEIT	} 5 Sekunden
15	3C		INR A	Laden des Akkus mit 02 (≙ gelb)
16	06 14		MVI B,14	} Laden von B mit 20 D,
18	CD 50 19		CALL ZEIT	} 2 Sekunden
1B	C3 E7 18		JMP ABF	Sprung zur Abfrage
1E	00 00		NOP	
1920	3E 00	GELB:	MVI A,00	Laden des Akkus mit 0 (≙ aus)
22	06 0A		MVI B,0A	} Laden von B mit 10 D,
24	CD 50 19		CALL ZEIT	} 1 Sekunde
27	C6 02		ADI 02	Laden des Akkus mit 02 (≙ gelb)
29	06 0A		MVI B,0A	} Laden von B mit 10 D,
2B	CD 50 19		CALL ZEIT	} 1 Sekunde
2E	C3 E7 18		JMP ABF	Sprung zur Abfrage

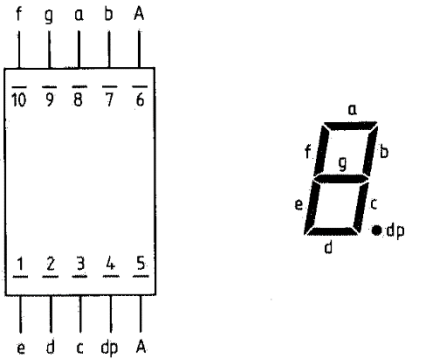
Programm zur Ampelsteuerung mit Vorwahl der Betriebsart.



Flußdiagramm zur Ampelsteuerung mit Vorwahl der Betriebsart über die Schalter an P1C0 und P1C1.

Übung 7: Ansteuerung einer 7-Segment-Anzeige

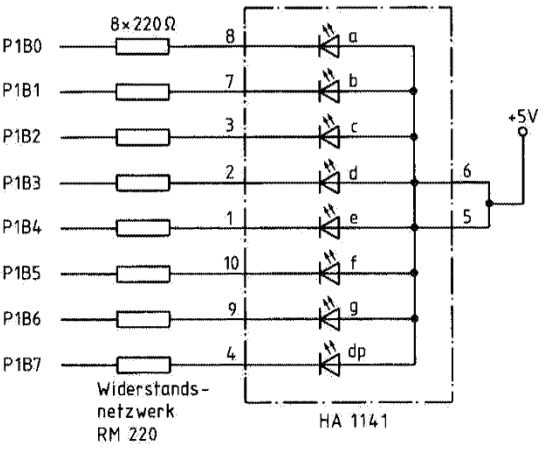
Die Dualzähler-Anzeige mit drei Leuchtdioden aus *Übung 5* soll nun erweitert werden durch eine *Ziffern-Anzeige.*
Dazu wird der Baustein HA 1141 verwendet, dessen Anschlußbelegung und Segmentzuordnung in der nachstehenden Abbildung dargestellt sind. Für die Ansteuerung soll der Port B des 8155 Nr. 1 verwendet werden (Port-Adresse FA, siehe *Bild 3.3.5*). Die Verbindung der Port-Anschlüsse mit den Segmenten des Anzeige-Bausteins HA 1141 ist dem Schaltplan zu entnehmen. Schließlich ist noch der Zusammenhang der Bit-Nr. des Ports bzw. der Wertigkeit des einzelnen Bits mit dem jeweils damit verbundenen Anzeige-Segment dargestellt.



Der 7-Segment-Anzeigebaustein *HA 1141*:
links ist die Anschlußbelegung, von oben gesehen, dargestellt und *rechts* die Anordnung der Segmente a, b, ..., g und des Dezimalpunktes dp auf der Vorderseite der Anzeige. A ist die gemeinsame Anode der Dioden.

Aus dem Schaltplan ist ersichtlich, daß ein Segment dann aufleuchtet, wenn der entsprechende Port-Anschluß einen 0-Wert führt, d. h. das entsprechende Bit 0 ist. Nachdem der Port B als Ausgabe-Port programmiert wurde (z. B. mit *Kommando 4*: Ausgabe von 02 H = 00000010 B auf die Steuerwort-Adresse F8), kann auf die Port-Adresse FA des Ports B des 8155 Nr. 1 nun irgendein Bitmuster ausgegeben werden.
Überlegen wir uns das Bitmuster für die Anzeige der Ziffer 0 auf dem Baustein HA 1141: Es müssen die Segmente a, b, c, d, e und f aufleuchten. Aus der Zuordnungstabelle für die Port-Bits ist zu ersehen, daß dazu die Bits 0 bis 5 den Wert null erhalten müssen; das Segment g und der Dezimalpunkt dp sollen dunkel bleiben, die Bits 6 und 7 werden daher mit 1 belegt:

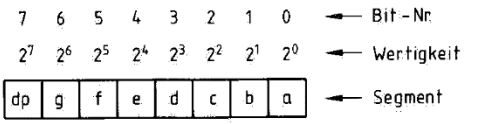
Segmente:	dp	g	f	e	d	c	b	a
Bitmuster:	1	1	0	0	0	0	0	0
Hex.-Code:		C				0		



Die Verschaltung des Ports B des 8155 Nr. 1 mit den Anzeige-Dioden des HA 1141. Die Nummern 1 bis 10 sind die Anschluß-Nummern des HA 1141.

Dieses Byte C0 ist, wie schon gesagt, auf die Port-Adresse FA auszugeben.
Soll die Ziffer 1 in der Anzeige erscheinen, so ergibt sich folgendes:

Dioden:	dp	g	f	e	d	c	b	a
Bitmuster:	1	1	1	1	1	0	0	1
Hex.-Code:		F				9		



Die Zuordnung der Bits des Ports B des 8155 Nr. 1 zu den Anzeigesegmenten. Es ist außerdem die Wertigkeit der Bits angegeben, damit das geforderte Ansteuermuster leichter in den Hex.-Code umgerechnet werden kann.

Noch einmal übersichtlich zusammengefaßt:
Mit *Kommando 4* ist zuerst der Baustein 8155 Nr. 1 zu initialisieren, so daß sein Port B Ausgabekanal ist, und dann ist das entsprechende Datenbyte auf die Port-Adresse FA auszugeben:

```
4. ↓
4.      PA  = F8 ↓
4.      F8  = 02 ↓   (Initialisierung)
4.      FA  = C0 ↓   (Anzeige der Ziffer 0)
4.      FA  = F9 ↓   (Anzeige der Ziffer 1)
usw.
```

Sie können auch einen Segmenttest mit einem kleinen Programm durchführen, indem Sie nacheinander an jedes Segment eine 0 legen und jeweils alle anderen mit Hilfe einer 1 dunkelsteuern. In der rechten Spalte finden Sie das Programm dazu.

Geben Sie dieses Programm ein und starten Sie es im *Kommando 7*. Wenn es richtig läuft, müssen die Segmente a, b, c, d, e, f und dp nacheinander für jeweils ca. ¼ Sekunde aufleuchten; anschließend bleibt das E für „Test-Ende" stehen.

Aufgaben

a)

Welche Ansteuermuster erfordert die Ausgabe der 16 Hexadezimal-Ziffern 0 bis F auf der 7-Segment-Anzeige?

b)

Schreiben Sie ein kleines Programm, das nacheinander auf Ihrer 7-Segmentanzeige ausgibt:
H
E
L
P

U
S

c)

Erweitern Sie Ihr Dualzähler-Programm aus *Übung 5* so, daß nach der dualen LED-Anzeige jeweils auch die zugehörige Ziffer an der 7-Segment-Anzeige ausgegeben wird.

Adresse	Hex.-Code	Labels	Mnemo-Code	
184D	31		LXI	SP,1C00
E	00			
F	1C			
50	3E		MVI	A,02
1	02			
2	D3		OUT	F8
3	F8			
4	3E		MVI	A,FE
5	FE			
6	D3		OUT	FA
7	FA			
8	CD		CALL	ZEIT
9	00			
A	19			
B	07		RLC	
C	FE		CPI	FE
D	FE			
E	C2		JNZ	1856
F	56			
60	18			
1	3E		MVI	A,86
2	86			
3	D3		OUT	FA
4	FA			
5	76		HLT	
1900	06	ZEIT:	MVI	B,FF
1	FF			
2	0E	NEU:	MVI	C,DB
3	DB			
4	0D	AB:	DCR	C
5	C2		JNZ	AB
6	04			
7	19			
8	05		DCR	B
9	C2		JNZ	NEU
A	02			
B	19			
C	C9		RET	

Lösungen

a)

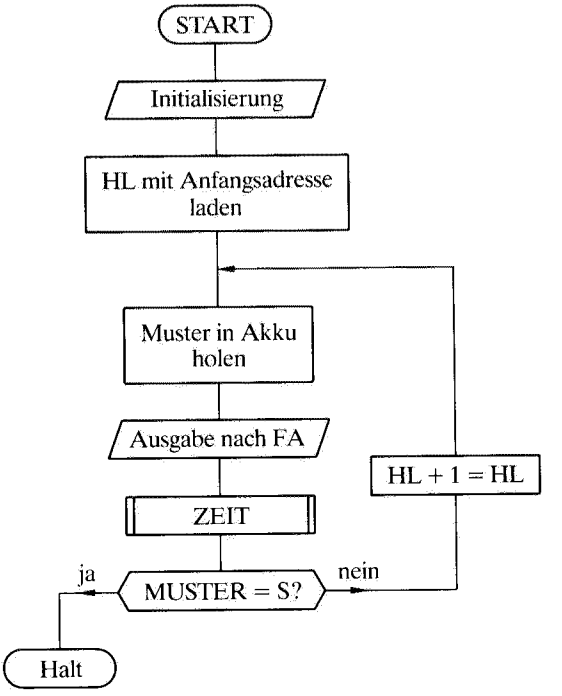
Anzeige	Bitmuster	Datenbyte, im Hex.-Code
	dpgfe dcba	
0	1100 0000	C0
1	1111 1001	F9
2	1010 0100	A4
3	1011 0000	B0
4	1001 1001	99
5	1001 0010	92
6	1000 0010	82
7	1111 1000	F8
8	1000 0000	80
9	1001 0000	90
A	1000 1000	88
B	1000 0011	83
C	1100 0110	C6
D	1010 0001	A1
E	1000 0110	86
F	1000 1110	8E

b)

Zuerst überlegen wir, wie die Bitmuster für die Buchstaben aussehen müssen:

	Bitmuster	Datenbyte, hex.
	dpgfe dcba	
H	1000 1001	89
E	1000 0110	86
L	1100 0011	C3
P	1000 1100	8C
	1111 1111	FF
U	1100 0001	C1
S	1001 0010	92

Wenn wir die Datenbytes für die Bitmuster in aufeinanderfolgende Speicherzellen schreiben, dann können wir sie indirekt adressiert über das Registerpaar HL innerhalb einer Schleife ausgeben.



Flußdiagramm für die Anzeige „HELP US". Am Schluß bleibt das S stehen.

b)

Adresse	Hex.-Code	Labels	Mnemo-Code
1850	31		LXI SP,1C00
1	00		
2	1C		
3	3E		MVI A,02
4	02		
5	D3		OUT F8
6	F8		
7	21		LXI H,1870
8	70		
9	18		
A	7E	AUS:	MOV A,M
B	D3		OUT FA
C	FA		
D	CD		CALL ZEIT
E	00		
F	19		
60	FE		CPI 92
1	92		
2	CA		JZ HALT
3	69		
4	18		
5	23		INX H
6	C3		JMP AUS
7	5A		
8	18		
9	76	HALT:	HLT
1870	89		
1	86		
2	C3		
3	8C		
4	FF		
5	C1		
6	92		

Programm für die Anzeige „HELP US". Das Unterprogramm ZEIT ist das gleiche wie das beim Segment-Test verwendete (ab Adresse 1900).

c)

Dualzählerprogramm-Erweiterung
Auch hier legen wir die 7-Segment-Ansteuermuster in aufeinanderfolgende Speicherplätze (ab 1840) ab und holen sie indirekt adressiert über das Registerpaar HL in den Akku.

Adresse	Hex.-Code	Mnemo-Code
1811	3E	MVI A,03
2	03	
3	D3	OUT F8
4	F8	
5	21	LXI H,1840
6	40	
7	18	
8	AF	XRA A
9	D3	OUT F9
A	F9	
B	57	MOV D,A
C	7E	MOV A,M
D	D3	OUT FA
E	FA	
F	23	INX H
20	7A	MOV A,D
1	3C	INR A
2	FE	CPI 08
3	08	
4	CA	JZ 1815
5	15	
6	18	
7	C3	JMP 1819
8	19	
9	18	

Die Muster für die Anzeige sind ab Adresse 1840 einzugeben:

Adresse	Inhalt
1840	C0
1	F9
2	A4
3	B0
4	99
5	92
6	82
7	F8

Dieses Programm kann – wie in *Übung 5* – nur im Einzelschritt abgearbeitet werden.

Übung 8: Erzeugen von Tönen

Entwickeln Sie auf dem ECB 85 ein Programm, das Ihnen in einer Endlosschleife über einen Lautsprecher einen Dreiklang vorspielt.
Benutzen Sie für die Erzeugung der Tonfrequenzen den Timer des Bausteins 8155 Nr. 2, mit dem Sie die Periodendauer auf 650 ns genau realisieren können (Timer-Adressen in *Bild 3.3.5*). Die Tonlängen erzeugen Sie mit programmierten Zeitschleifen. Die Frequenzen der zu erzeugenden Töne des C-Dur-Dreiklangs sind:

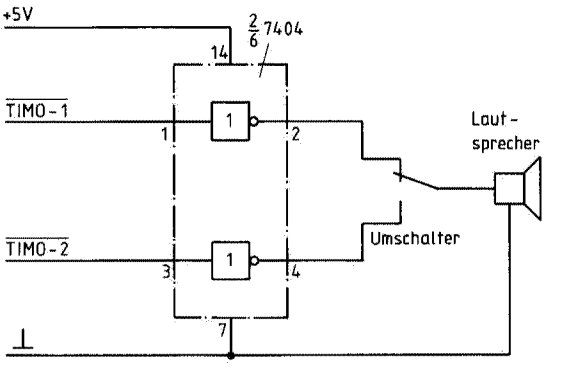
c = 261 Hz,
e = 329 Hz,
g = 392 Hz.

Anleitung:

Die Programmierung des Timers im Baustein 8155 wurde in *Kapitel 3.3* ausführlich behandelt. In *Bild 3.3.9* wurden die Betriebsarten gezeigt. Für die hier gewünschte Tonerzeugung ist die Betriebsart 1 geeignet. Bei ihr liefert der Timerausgang einen Rechteckpuls mit dem Tastverhältnis 2, d. h. die Periodendauer ist 2mal so lang wie die Impulsdauer; mit anderen Worten: Impulse und Impulspausen sind gleich lang. Die Periodendauer ist gleich dem zuvor eingegebenen Timerwert (14 Bits, siehe *Bild 3.3.7*), multipliziert mit der Taktperiode, die am Timereingang anliegt. Dies ist beim Timer des 8155 Nr. 1 im ECB 85 die Periode des Signals CLK vom 8085, also 325 ns (Frequenz: 3,072 MHz). Soll der Timer des 8155 Nr. 2 benutzt werden, so ist sein Timereingang TIMI-2 (siehe *Bild 3.4.1*) mit CLK zu verbinden.
Wird also beispielsweise der Timer mit dem Zahlenwert tausend geladen, dann liefert der Timerausgang in Betriebsart 1 einen Rechteckpuls mit der Frequenz 3,072 kHz. Gibt man diesen Rechteckpuls auf einen Lautsprecher, so kann man den Ton mit der genannten Frequenz hören und zusätzlich Obertöne mit allen ungeradzahligen Vielfachen der Grundfrequenz (Stichwort: Fourier-Analyse des Rechteckpulses).

Zur Hardware:

Es ist eine Schaltung aufzubauen, wie sie in der nachstehenden Abbildung dargestellt ist: Der Lautsprecher kann (über Negationsglieder) wahlweise mit dem Timerausgang des 8155 Nr. 1 ($\overline{\text{TIMO-1}}$) oder des 8155 Nr. 2 ($\overline{\text{TIMO-2}}$) verbunden werden. Zusätzlich muß – wie schon gesagt – TIMI-2 mit CLK verbunden werden! Der 6fach-Inverter-Baustein 7404 ist in *Übung 5* vorgestellt worden.



Erzeugen anderer Töne:

Der Vollständigkeit halber seien hier noch die Frequenzen der anderen vier Töne der eingestrichenen Oktave angegeben:

d = 293 Hz
f = 349 Hz
a = 440 Hz („Kammerton a")
h = 494 Hz

Durch Verdoppeln bzw. Halbieren dieser Frequenzen können Töne der darüber- bzw. darunterliegenden Oktave erzeugt werden. Damit ist es Ihnen möglich, außer dem einfachen Dreiklang, der im folgenden realisiert wird, auch Melodien mit dem ECB 85 zu erzeugen. Die evtl. dazu nötigen Pausen werden folgendermaßen programmiert: Der Timer ist zu stoppen (vgl. dazu *Bild 3.3.6:* Ausgabe des Steuerwortes 01XX XXXX an das Steuerwort-Register) und dann die entsprechende Zeitspanne per Unterprogramm abzuwarten.

Nun zum Programm für den c-e-g-Dreiklang:

Am Beispiel des Tons c mit der Frequenz 261 Hz wird hier ausführlich gezeigt, wie er erzeugt wird; für die anderen Töne wird dann nur kurz das Ergebnis angegeben.

Erster Schritt: Berechnen des Timerwertes aus der Frequenz

Die Frequenz ist hier 261 Hz, also beträgt die Periodendauer (1/261) s = 3,83142 ms. Weil 1 Periode des Signals CLK eine Dauer von 325 ns hat, müssen (3,83142 ms)/(325 ns) = 11788 Perioden mit je 325 ns ablaufen, um die geforderte Dauer von 3,83142 ms zu realisieren. Der Timer ist also mit 11788 zu laden. Dieser Wert ist in das hexadezimale Zahlensystem umzurechnen, und es ergibt sich: 2E0C. Die beiden Timer-Bytes lauten also binär geschrieben: 0010 1110 0000 1100. Dazu müssen aber noch in den beiden höchstwertigen Stellen die Betriebsart-Bits M_2 und M_1 eingesetzt werden; siehe hierzu *Bild 3.3.7.* Wie schon erwähnt, brauchen wir die Betriebsart 1, also ist $M_2 = 0$ und $M_1 = 1$, und die Timer-Bytes lauten binär: 0110 1110 0000 1100 bzw. hexadezimal: 6E0C – und hiermit ist der Timer zu laden (Adressen F5 bzw. F4 für den 8155 Nr. 2, siehe *Bild 3.3.5*).

Zweiter Schritt: Starten des Tons

Hierzu muß nach dem Laden des Timers das Steuerwort 11XX XXXX an das Steuerwort-Register des 8155 Nr. 2 mit der Adresse F0 ausgegeben werden (vgl. dazu *Bild 3.3.6*). Z. B. wird 1100 0000 B = C0 H ausgegeben; nun erklingt der Ton so lange, bis mit Ausgabe eines neuen Steuerwortes der Timer aus diesem Betriebszustand herausgeholt wird.

Dritter Schritt:

Die Dauer des Tons wird mit Hilfe einer Zeitschleife bestimmt, wie sie in den vorigen Übungen schon mehrfach verwendet wurde. Darauf muß hier nicht mehr näher eingegangen werden.

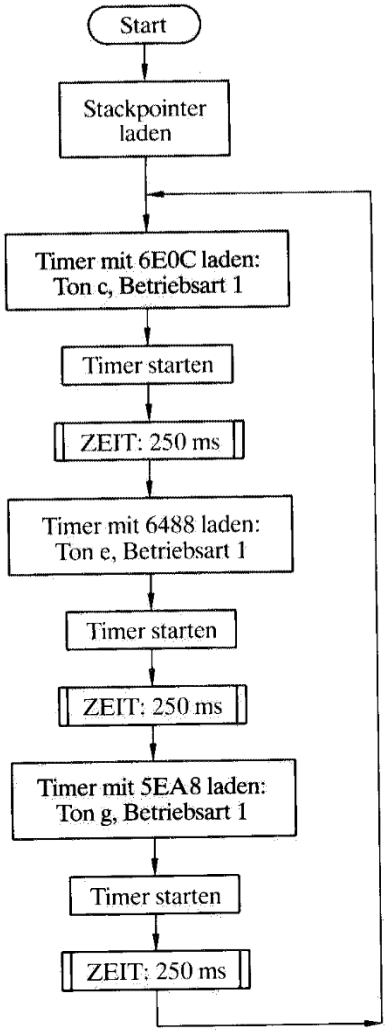
Vierter Schritt:

Weitere Töne werden nach Ablauf der Zeitschleife folgendermaßen erzeugt: Die Timerbytes werden neu geladen, und mit der darauf folgenden Ausgabe des Steuerwortes C0 an das Steuerwort-Register (Adresse F0) erklingt der neue Ton, der dann wiederum so lange erklingt, bis die Zeitschleife abgelaufen ist. Aus dem weiter unten wiedergegebenen Flußdiagramm für den Dreiklang ist dies gut zu ersehen.

Berechnung der Timerwerte für die anderen Töne:

Das einzige, was zum Erstellen des Dreiklang-Programms nun noch fehlt, ist die Umrechnung der anderen Tonfrequenzen in die Timerwerte. Diese geschieht genau nach dem beim Ton c gegebenen Verfahren.

Ton e: Frequenz: 329 Hz; Periodendauer: (1/329) s = 3,03951 ms. Darin sind 325 ns 9352mal enthalten; dies in das Hex.-System umgerechnet ergibt: 2488 H, im Binär-System: 0010 0100 1000 1000. Da-



Flußdiagramm für den Dreiklang.

zu kommen noch die Bits 0 und 1 für M_2 und M_1 (= Addition von 4 in der höchstwertigen Hex.-Stelle); der Timer ist hier also mit 6488 H zu laden.
Ton g: Frequenz 392 Hz, Periodendauer 2,55102 ms = 7848 * 325 ns. Timerwert demnach (dezimal) 7848; das ist 1EA8 H, mit den Betriebsart-Bits also: 5EA8 H.

Adresse	Hex.-Code	Mnemo-Code
1900	31	LXI SP,1C00
1	00	
2	1C	
3	3E	MVI A,0C
4	0C	
5	D3	OUT F4
6	F4	
7	3E	MVI A,6E
8	6E	
9	D3	OUT F5
A	F5	
B	3E	MVI A,C0
C	C0	
D	D3	OUT F0
E	F0	
F	CD	CALL ZEIT
10	40	
1	19	
2	3E	MVI A,88
3	88	
4	D3	OUT F4
5	F4	
6	3E	MVI A,64
7	64	
8	D3	OUT F5
9	F5	
A	3E	MVI A,C0
B	C0	
C	D3	OUT F0
D	F0	
E	CD	CALL ZEIT
F	40	
20	19	
1	3E	MVI A,A8
2	A8	
3	D3	OUT F4
4	F4	
5	3E	MVI A,5E
6	5E	
7	D3	OUT F5
8	F5	
9	3E	MVI A,C0
A	C0	
B	D3	OUT F0
C	F0	
D	CD	CALL ZEIT
E	40	
F	19	
30	C3	JMP 1903
1	03	
2	19	

Programm für den Dreiklang.

Adresse	Hex.-Code	Label	Mnemo-Code
1940	0E	ZEIT:	MVI C,FA
1	FA		
2	16		MVI D,DB
3	DB		
4	15		DCR D
5	C2		JNZ 1944
6	44		
7	19		
8	0D		DCR C
9	C2		JNZ 1942
A	42		
B	19		
C	C9		RET

Unterprogramm ZEIT.

Erweiterung der Übung 8

Es ist ein Programm zu erstellen, das eine Melodie ausgibt (z. B. „Bruder Jakob") und diese nach einer kurzen Pause immer wieder neu spielt. Als Hardware-Schaltung wird der gleiche Lautsprecher-Anschluß verwendet, wie er eben beschrieben wurde.

Abspeicherung der Melodie:

Die zum Laden des Timers für jeden Ton benötigten Timer-Bytes (s. o.) werden in aufeinanderfolgende Speicherplätze, z. B. beginnend bei 1A00, abgelegt. Dabei gehören immer zwei Bytes zu einem Ton, da der Timer des 8155 ja mit zwei Bytes zu laden ist. Am Ende der Melodie müssen zur Erzeugung der Pause mehrere Byte-Paare 40, 00 hintereinander eingegeben werden; damit wird der Timer angehalten. Danach folgt ein einzelnes Byte 00 (in einer Speicherzelle mit geradzahliger Adresse!); dies dient dazu, nach dem Ende der Pause wieder an den Anfang der Melodie zu springen.

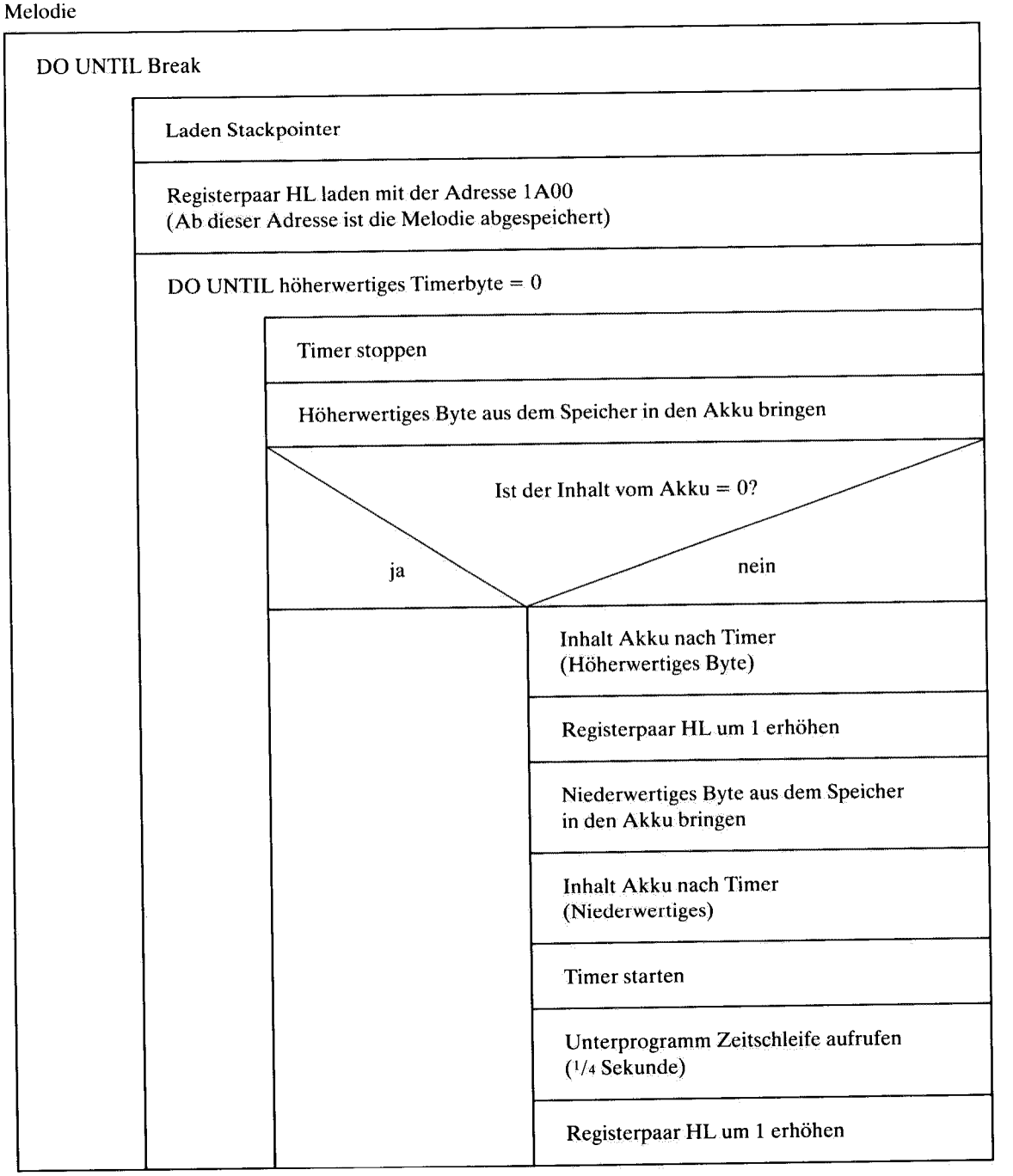
Prinzip des Programms:

Nach dem Laden des Stackpointers wird das Registerpaar H,L mit der Adresse des ersten Melodie-Bytes geladen. Nun beginnt die Hauptschleife (SCHLEI), in der zunächst der Timer angehalten wird. Danach wird das erste (höherwertige) Melodie-Byte in den Akku geholt und darauf geprüft, ob es null ist (wenn ja, Beginn von Anfang an). Danach wird dieses erste Melodie-Byte an den Timer ausgegeben und der Inhalt des Registerpaares H,L inkrementiert, so daß jetzt das zweite Melodie-Byte eingeholt und an den Timer ausgegeben werden kann. Nun ist nur noch der Timer zu starten und die Zeitschleife aufzurufen, bis nach nochmaligem Inkrementieren von H,L das Spiel mit dem nächsten Ton weitergehen kann.

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
1900	31	ANF:	LXI SP,1C00	Stackpointer laden
1	00			
2	1C			
3	26		MVI H,1A	} Anfangsadresse der Melodie
4	1A			in Registerpaar HL bringen
5	2E		MVI L,00	
6	00			
7	3E	SCHLEI:	MVI A,40	} Timer anhalten
8	40			(Timer 0)
9	D3		OUT F0	
A	F0			
B	7E		MOV A,M	Höherwertiges Melodie-Byte in den Akku
C	C6		ADI 0	} Kontrolle, ob Akku-Inhalt = 0
D	00			(dann wird hier Zero-Bit gesetzt)
E	CA		JZ ANF	Nach Ende der Melodie neu anfangen
F	00			
10	19			
1	D3		OUT F5	Höherwertiges Byte in Timer
2	F5			
3	23		INX H	Registerpaar HL zeigt jetzt auf nächstes Melodie-Byte
4	7E		MOV A,M	} Niederwertiges Timerbyte lesen und
5	D3		OUT F4	in Timer laden
6	F4			
7	3E		MVI A,C0	} Timer starten
8	C0			
9	D3		OUT F0	
A	F0			
B	CD		CALL ZEIT	
C	40			
D	19			
E	23		INX H	Registerpaar HL zeigt auf nächstes Melodie-Byte
F	C3		JMP SCHLEI	Nächster Durchlauf der Schleife soll
20	07			beginnen, (nächster Ton).
1	19			

Programm für einen Melodie-Ablauf.

Melodie



Struktogramm für einen Melodie-Ablauf.

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
1940	0E	ZEIT:	MVI C,F9	Zeitschleife von 1/4 Sekunde
1	F9			
2	16	NEU:	MVI D, DB	
3	DB			
4	15	AB:	DCR D	
5	C2		JNZ AB	
6	44			
7	19			
8	0D		DCR C	
9	C2		JNZ NEU	
A	42			
B	19			
C	C9		RET	Zurück zur Hauptschleife

Unterprogramm ZEIT

Adresse	Inhalt	Adresse	Inhalt	Adresse	Inhalt
1A00	5E	1A1E	52	1A3C	69
1	A9	F	3B	D	05
2	5B	1A20	54	E	5E
3	51	1	76	F	A9
4	58	2	56	1A40	40
5	54	3	FC	1	00
6	5E	4	58	2	40
7	A9	5	54	3	00
8	5E	6	5E	4	40
9	A9	7	A9	5	00
A	5B	8	54	6	40
B	51	9	76	7	00
C	58	A	52	8	40
D	54	B	3B	9	00
E	5E	C	54	A	40
F	A9	D	76	B	00
1A10	58	E	56	C	00
1	54	F	FC		
2	56	1A30	58		
3	FC	1	54		
4	54	2	5E		
5	76	3	A9		
6	58	4	5E		
7	54	5	A9		
8	56	6	69		
9	FC	7	05		
A	54	8	5E		
B	76	9	A9		
C	54	A	5E		
D	76	B	A9		

Timerbytes für die Melodie „Meister Jakob".

Übung 9: Geschwindigkeits-Messung

In den vorangegangenen Übungen wurden Informationen von der Peripherie eingeholt sowie Signale dorthin ausgegeben, und es wurden Zeitspannen bestimmter Länge realisiert. Dies wird in dieser Übung angewendet und vertieft, wobei nun auch eine Zeitspanne zu *messen* ist.

Auf einer Modell-Rennstrecke passiert ein Modellauto zwei Lichtschranken, die 20 cm Abstand voneinander haben. Die durchschnittliche Geschwindigkeit des Modellautos entlang dieser Meßstrecke ist mit Hilfe des ECB 85 zu bestimmen und auf einer 4stelligen Anzeige auszugeben.

Es muß also zunächst die Zeitspanne t zwischen den beiden Lichtschranken-Signalen (s. u.) gemessen werden. Da die Wegstrecke s bekannt ist (20 cm), kann die Geschwindigkeit v durch Division berechnet werden: v = s/t. Schließlich muß der im Computer binär vorliegende Wert für die 4stellige dezimale Anzeige aufbereitet werden.

Die Übung gliedert sich in zwei Teile:

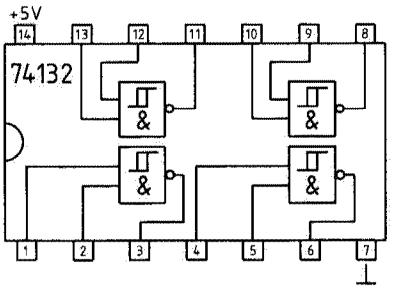
1. Hardware: Aufbau und Test der Anwender-Schaltung und der Anzeige;
2. Software: Erstellen des Programms.

Hardware

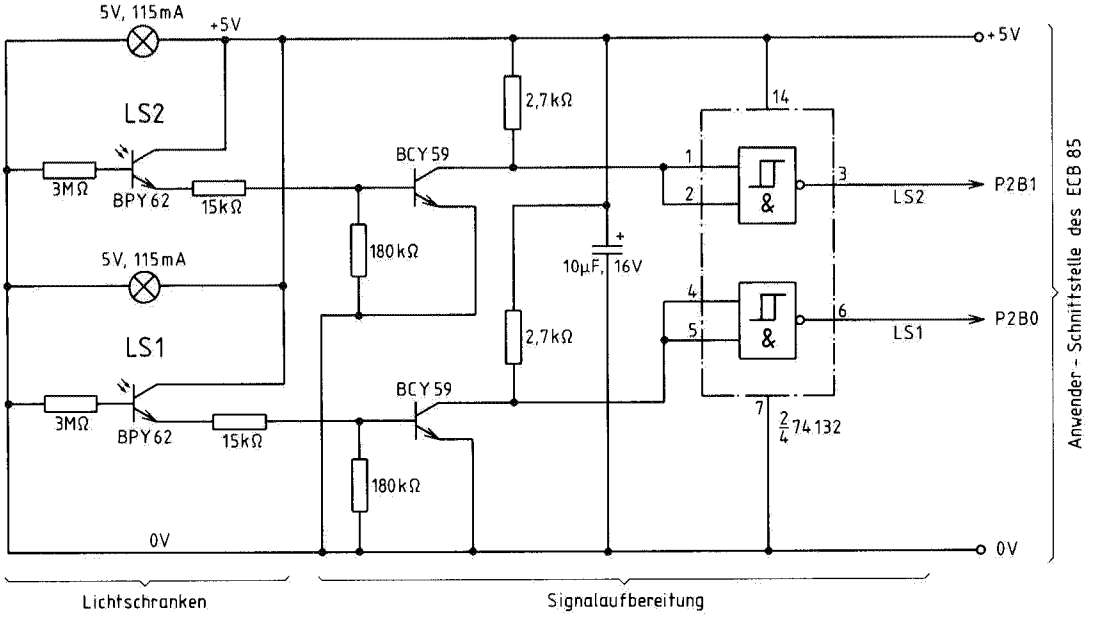
a) Lichtschranken-Signale

Für die Verbindung des ECB 85 mit der Peripherie wird in dieser Übung der E/A-Baustein 8155 Nr. 2 verwendet. Zwei Anschlüsse seines Ports B werden mit den Lichtschranken-Signalen LS1 und LS2 belegt:

P2B0 mit LS1 und P2B1 mit LS2.

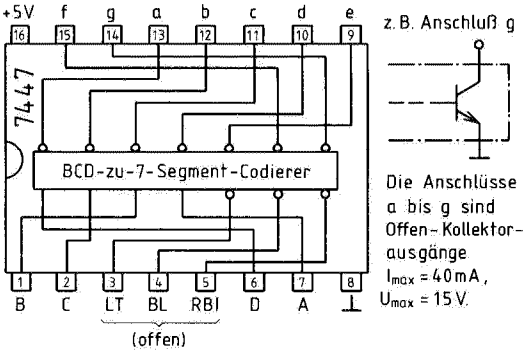


Anschlußplan (von oben gesehen) des Bausteins 74132, der vier NAND-Schmitt-Trigger-Glieder enthält.



Anwender-Schaltung zur Geschwindigkeitsmessung.

Die Ports A und B des 8155 Nr. 1 dienen zum Ausgeben der vier Anzeigestellen (s. u.).
Jede Lichtschranke wird aufgebaut aus einem kleinen Lämpchen und einem Fototransistor mit Arbeitswiderständen; dies ist im Schaltplan dargestellt. Aus ihm geht weiterhin die Schaltung zur Signalaufbereitung hervor: Jedes Lichtschranken-Signal wird über eine Transistor-Verstärkerstufe auf ein invertierendes Schmitt-Trigger-Glied (im IC 74132, siehe Anschlußplan) geführt, dessen Ausgang dann mit dem entsprechenen Port-Anschluß P2B0 bzw. P2B1 des ECB 85 zu verbinden ist. Natürlich muß die ganze Schaltung auch mit dem 0-V-Anschluß (Masse) des ECB 85 verbunden werden; die ECB-85-Anwender-Schnittstelle (vgl. *Bild 3.4.1*) liefert auch die 5-V-Spannungsversorgung für diese Schaltung. *Wichtig:* der Elektrolyt-Kondensator (10μF/16V) muß mit möglichst kurzer Leitungsführung zu den Anschlüssen 14 und 7 des ICs 74132 in die Schaltung eingebaut werden.
Wird eine Lichtschranke unterbrochen, d. h. erhält der Fototransistor vom Lämpchen kein Licht, dann führt der Ausgang LS1 bzw. LS2 der Anwender-Schaltung L-Pegel. Wenn das Modellauto eine Lichtschranke passiert (oder wenn diese auf andere Weise verdunkelt wird), so ergibt das also einen 0-Wert des entspr. Port-Bits P2B0 bzw. P2B1.



Anschlußplan des BCD-zu-7-Segment-Codierers 7447. Die Bausteine 7446 und 74247 sind hierzu pin-kompatibel, d. h. sie haben die gleiche Anschlußbelegung. Die Ausgänge a, b, c, . . ., g sind über 220-Ω-Widerstände mit den Anzeigedioden des Bausteins HA 1141 (siehe *Übung 7*) zu verbinden.

b) Anzeige

Wie schon erwähnt, werden für die Ausgabe der anzuzeigenden Geschwindigkeits-Werte die Ports A und B des 8155 Nr. 1 benutzt: P1A und P1B. Die Anzeige besteht aus vier 7-Segment-Anzeigebausteinen des Typs HA 1141, der schon in *Übung 7* eingesetzt wurde; dort wurden aber 8 Bits eines Ports (über Widerstände) direkt mit den 8 Anzeigedioden (einschl. Dezimalpunkt) des HA 1141 verbunden.

Anzeige:	Eingänge D	C	B	A	Zahlenwert
	Wertigkeit				
Zeichen	2^3	2^2	2^1	2^0	hex.	dez.
0	0	0	0	0	0	0
1	0	0	0	1	1	1
2	0	0	1	0	2	2
3	0	0	1	1	3	3
4	0	1	0	0	4	4
5	0	1	0	1	5	5
*6	0	1	1	0	6	6
7	0	1	1	1	7	7
8	1	0	0	0	8	8
**9	1	0	0	1	9	9
c	1	0	1	0	A	10
ɔ	1	0	1	1	B	11
u	1	1	0	0	C	12
c	1	1	0	1	D	13
t	1	1	1	0	E	14
(dunkel)	1	1	1	1	F	15

*6 beim 74247

**9 beim 74247

Die vom BCD-zu-7-Segment-Codierer 7447 repräsentierten Zeichen in Abhängigkeit von den Signalen an den vier BCD-Eingängen A, B, C und D.

Hier soll aber anders vorgegangen werden: Wenn man die Umcodierung aus den 8 Bits des Ports in das 7-Segment-Anzeigemuster nicht per Programm (wie in *Übung 7*) vornimmt, sondern mit Hilfe des TTL-Bausteins 7447 (s. u.), so können die 8 Bits (1 Byte) des Ports in eine zweistellige Anzeige (zwei Halbbytes) umgesetzt werden. Die Abbildungen zeigen hier den Anschlußplan des BCD-zu-7-Segment-Codierers (eigentlich 4-Bit-zu-7-Segment-Codierers) 7447 sowie die darzustellenden Zeichen in Abhängigkeit vom Bitmuster an seinen Eingängen A, B, C und D mit den Wertigkeiten 2^0, 2^1, 2^2 und 2^3. Vom BCD-Code spricht man beim Umsetzen der Zahlenwerte 0 bis 9 in den Dual-(Binär-)Code.

Die Verschaltung von 4 Bits eines 8155-Ports für eine Anzeige-Stelle ist ebenfalls im Bild dargestellt, und schließlich auch die Verschaltung aller vier Anzeigestellen. Der Port A dient zur Anzeige der beiden höherwertigen Ziffern des Geschwindigkeits-Wertes (also der ganzen km/h), und der Port B zur Anzeige der Zehntel und Hundertstel km/h.

c) *Test der Schaltungen*

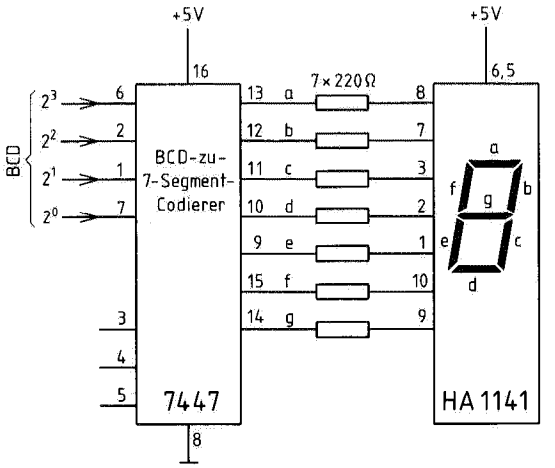
Gemäß den vorstehenden Abbildungen sind die Port-Anschlüsse P1A0–P1A7 und P1B0–P1B7 sowie P2B0 und P2B1 mit der Anwender-Schaltung zu verbinden (Anschlüsse +5 V und Masse nicht vergessen!). Sollte nun das Monitorprogramm des ECB 85 nicht weiterlaufen, ist die Schaltung auf fehlerhafte Kontakte zu prüfen. Wenn das Monitorprogramm läuft, dann sind nacheinander die folgenden Prüfschritte auszuführen:

1. Prüfen der Anzeige: Bausteine 8155 Nr. 1 und 2 initialisieren, d. h. mit Kommando 4 den Wert 03 auf die Adresse F8 ausgeben: Port 1A und 1B dienen zur Ausgabe, und den Wert 00 auf die Adresse F0: Port 2B zur Eingabe; vgl. *Bild 3.3.5 und 3.3.6.* Nach Abschluß der Initialisierung müssen die vier Anzeigestellen jeweils eine Null zeigen. Geben Sie nun auf die Adressen F9 und FA jeweils den Wert 88 aus. Es müssen jetzt alle Anzeigedioden leuchten: Anzeige 8888. Ist dies nicht der Fall, dann legen Sie an den Anschluß jedes dunklen Segments 0 V an. Wenn es dann leuchtet, liegt ein Anschlußfehler vor; bleibt es jedoch dunkel, so ist die Anzeige defekt und muß ausgetauscht werden.

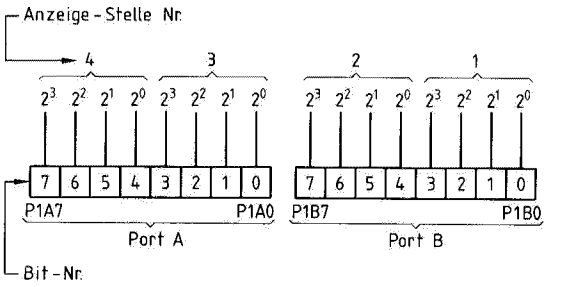
2. Prüfen der Lichtschranken-Signale: Mit Kommando 5 wird der Port 2B (Adresse F2) abgefragt, an dessen Bits 0 und 1 die Lichtschranken-Signale LS1 und LS2 anliegen (siehe Schaltplan).

Sind beide Lichtschranken offen (hell), so muß das Bitmuster xxxx xx11 resultieren (hex. z. B. C3).

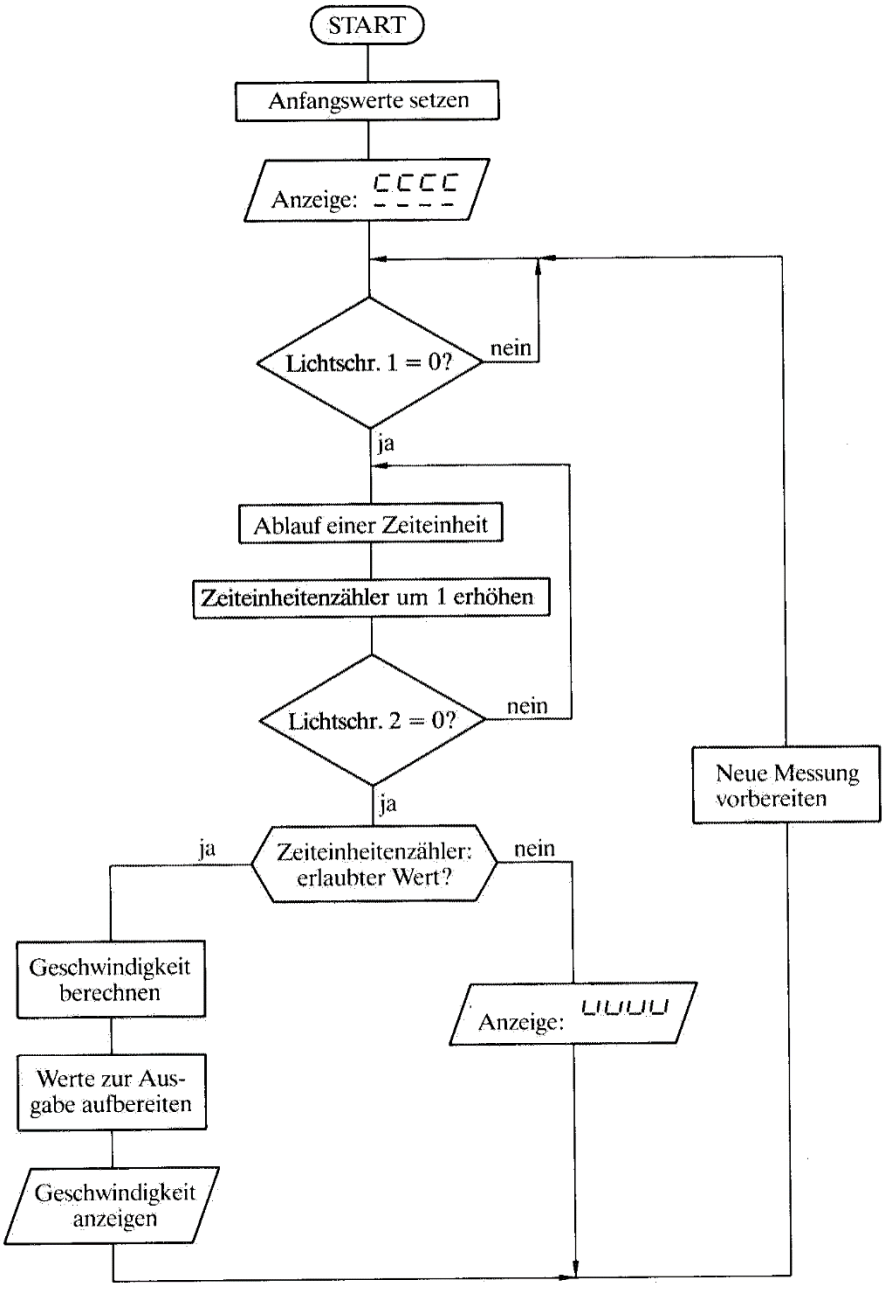
Beim Verdunkeln von LS1 muß sich mit Kommando 5 das Bitmuster xxxx xx10 ergeben (hex. z. B. C2).



Verschaltung eines Codierers 7447 mit dem Baustein HA 1141 für eine Anzeige-Stelle. Zum Anschlußplan des HA 1141 siehe *Übung 7.* Die Anschlüsse +5 V und Masse sind mit den entspr. Anschlüssen der Anwender-Schnittstelle des ECB 85 zu verbinden.



Die Verschaltung der Ports A und B des 8155 Nr. 1 für die vier dezimalen Anzeige-Stellen. Die Anschlüsse 2^3,2^2,2^0,2^1 sind jeweils die BCD-Eingänge des Codierer-Bausteins 7447 (siehe vorige Abb.). Die Umsetzung des 8-Bit-Dual-Codes eines jeden Ports in den jeweils 2stelligen BCD-Code für die Geschwindigkeits-Anzeige geschieht per Programm; siehe Abschnitt „Software" dieser Übung. Die Lage der Port-Anschlüsse auf der Anwender-Schnittstelle des ECB 85 geht aus *Bild 3.4.1* hervor.



Flußdiagramm zur Geschwindigkeitsmessung.

Beim Verdunkeln vom LS2 ergibt eine erneute Abfrage mit Kommando 5 das Bitmuster xxxx xx01 (hex. z. B. C1).
Wenn sich hier für die niedrigstwertigen beiden Bits des Ports B (68155 Nr. 2) andere Werte ergeben sollten, ist die Anwender-Schaltung entweder im Lichtschranken- oder im Verstärker-Teil fehlerhaft.

Software

Das zu erstellende Programm soll folgende Abläufe realisieren:
1. Nach dem Start des Programms soll ⊏⊏⊏⊏ in der Anzeige erscheinen, bis das low-Signal von LS1 eintrifft.
2. Nun soll die Zeitmessung beginnen, und nach dem Eintreffen des low-Signals von LS2 soll zunächst geprüft werden, ob die Geschwindigkeit im erlaubten Bereich, nämlich zwischen 1 und 99,99 km/h, liegt. Ist das nicht der Fall, dann soll ⊔⊔⊔⊔ auf der Anzeige erscheinen und eine neue Messung möglich sein.
3. Liegt die Geschwindigkeit im erlaubten Bereich, so ist sie zunächst zu berechnen und dann 4stellig dezimal auf der Anzeige auszugeben.
4. Danach soll das Programm für eine weitere Messung bereit sein.

Anleitung zur Lösung

Die zulässigen Geschwindigkeits-Werte sollen – wie gesagt – zwischen 1,00 und 99,99 km/h liegen. Gemäß der Gleichung

$$v = \frac{s}{t} = \frac{20\text{ cm}}{t} = \frac{720}{t\,(\text{ms})}\ (\text{km/h})$$

bedeutet dies, daß die zwischen den Lichtschranken-Signalen verstrichene Zeitspanne eine Dauer von 7,2 ms (bei 100 km/h) bis 720 ms (bei 1 km/h) hat. Wählt man als Zeiteinheit 1 ms, so braucht man als Zeiteinheiten-Zähler ein Registerpaar, weil das eine Byte eines einzelnen Registers nur Werte bis zu 255

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
1800	3E 03		MVI A,03	Initialisieren
	D3 F8		OUT F8	P1A, P1B: Ausgabe; P1C: Eingabe
	31 00 1C		LXI SP, 1C00	SP ← 1C00
	3E DD		MVI A,DD	} Anzeige: ⊏⊏⊏⊏
	D3 F9		OUT F9	
	D3 FA		OUT FA	
180D	11 00 00	NEU:	LXI D,0000	DE ← 0
1810	DB F2	SCHL1:	IN F2	} Abfrage LS1, ob dunkel, wenn nicht: zurück nach SCHL1
	E6 01		ANI 01	
	C2 1018		JNZ SCHL1	
1817	3E FF	SCHL2:	MVI A,FF	} Löschen der Anzeige
	D3 F9		OUT F9	
	D3 FA		OUT FA	
	13		INX D	Zählregister DE + 1
	37		STC	} Carry garantiert auf Null (≙ 8 Takte)
	3F		CMC	
1820	21 DF E3		LXI H, E3DF	} Abfrage, ob Zähler ≤ 7200, sonst Fehlerausgabe
	19		DAD D	
	DA 6E 18		JC FEHL	
	06 0B		MVI B, 0B	
1829	7F	ZEIT:	MOV A,A	
	05		DCR B	Zeitschleife ≙ 213 Takte
	C2 29 18		JNZ ZEIT	

Beginn des Programms zur Geschwindigkeitsmessung.

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
	C6 00		ADI 0	
	7F		MOV A,A	
	DB F2		IN F2	} Abfrage LS 2,
	E6 02		ANI 02	} ob dunkel; wenn nicht:
	C2 17 18		JNZ SCHL2	} zurück nach SCHL2
1838	7A		MOV A,D	}
	FE 00		CPI 00	}
	C2 4418		JNZ X1	} Abfrage ob Zähler (≙ D E) > 72,
	7B		MOV A,E	} sonst Fehlerausgabe
	FE 48		CPI 48H	}
	DA 6E18		JC FEHL	}
1844	D5	X1:	PUSH D	D E ≙ Zae = > Stack
	EB		XCHG	H L ≙ Zae
	1120 1C		LXI D, 1C20	D E mit 7200 geladen
	CD 00 1C		CALL DIV	7200 ÷ Zae
184C	E5		PUSH H	Rest nach Stack gerettet
	CD 0019		CALL AUSG	
	D3 F9		OUT F9	Ausgabe dezimal an F9 (P1A)
	D1		POP D	D E ≙ Rest
	06 02		MVI B, 02	}
1855	21 0000	X2:	LXI H, 0000	}
	19		DAD D	}
	29		DAD H	} Multipliziere den Rest mit 100
	29		DAD H	} (2malige Mult. mit 10). Danach:
	19		DAD D	} D E ≙ Rest * 100
	29		DAD H	}
	EB		XCHG	}
	05		DCR B	}
	C2 5518		JNZ X2	}
1862	E1		POP H	H L ≙ Zae
	CD 00 1C		CALL DIV	100 * Rest ÷ Zae
	CD 00 19		CALL AUSG	
	D3 FA		OUT FA	Ausgabe dezimal an FA (P1B)
	C3 0D 18		JMP NEU	
186E	3E CC	FEHL:	MVI A,CC	} Fehlerausgabe:
	D3 F9		OUT F9	} Anzeige ⊔⊔⊔⊔
	D3 FA		OUT FA	}
	C3 0D18		JMP NEU	
1900	21 0A 00	AUSG:	LXI H, 000A	H L ≙ 10
	CD 00 1C		CALL DIV	Quotient ÷ 10
	7B		MOV A,E	
	07		RLC	}
	07		RLC	} Verschieben des Akkuinhaltes um
	07		RLC	} 4 Bit nach links
	07		RLC	}
	85		ADD L	Rest der Div. durch 10 in niederwertiges Akku-Halbbyte
190C	C9		RET	

(hex. FF) enthalten kann. Verwendet man also ein Registerpaar (z. B. DE) als Zähler, dann kann man sogar, um genauer messen zu können, eine Zeit von 0,1 ms als Zeiteinheit vorsehen; dann liegen die erlaubten Werte zwischen 72 und 7200, und die Gleichung lautet:

$$v = \frac{7200}{t\,(0{,}1\text{ ms})}\ (\text{km/h})$$

Im Programm ist nach dem Eintreffen des LS1-Signals zunächst abzufragen, ob die verstrichene Zeit nicht zu lang ist. Nach dem Eintreffen des LS2-Signals ist zu prüfen, ob die Zeit nicht zu kurz war. Das Flußdiagramm zeigt im groben den Programmablauf.

Für die Berechnung der Geschwindigkeit und für die Umrechnung in eine 4stellige Dezimalzahl wird ein Divisionsprogramm als Unterprogramm benötigt; siehe Programm ab Adresse 1C00. Es wird mit CALL DIV aufgerufen (im Hex.-Code: CD 00 1C) und dividiert die beiden im Registerpaar DE stehenden Bytes durch die beiden in HL befindlichen Bytes. Nach der Division steht der (natürlich ganzzahlige) Quotient in DE und der Rest in HL.

Zur Berechnung der Geschwindigkeit wird, falls der Zähler einen erlaubten Wert enthält, zunächst die Zahl 7200 durch den nach HL gebrachten Inhalt des Zeiteinheiten-Zählers, d. h. durch die Anzahl der verstrichenen Zehntel Millisekunden, dividiert (siehe Gleichung). Danach steht der ganzzahlige Geschwindigkeits-Wert im Registerpaar DE (binär!) und der Rest in HL.

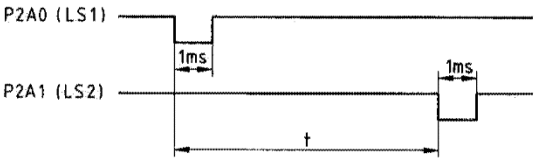
Das Unterprogramm AUSG dient nun dazu, die Ausgabe des ganzzahligen Geschwindigkeits-Wertes vorzubereiten: Der in DE stehende Wert wird durch 10D (= 0AH) dividiert, so daß sich danach in DE (genauer: in E) die höherwertige und in HL (genauer: in L) die niederwertige Dezimalstelle befindet. Nun wird der Inhalt von E in den Akku gebracht. Durch Verschieben des Akku-Inhalts um vier Stellen nach links und Addieren des Inhalts von L hierzu stehen schließlich am Ende des Unterprogramms AUSG die beiden Dezimalstellen des ganzzahligen Geschwindigkeits-Wertes im BCD-Code in den beiden Halbbytes des Akkus und können so direkt auf den Port A (Adresse F9) ausgegeben werden. Der Rest der ersten Division wird mit hundert multipliziert (dies liefert die Nachkomma-Stellen des Geschwindigkeits-Wertes) und im Unterprogramm AUSG ebenso wie der ganzzahlige Teilwert behandelt; Ausgabe dann auf Port B, Adresse FA.

Adresse	Hex.-Code	Labels	Mnemo-Code
1C00	22 401C	DIV:	SHLD 1C40
1C03	01 0000		LXI B,0000
1C06	C5		PUSH B
1C07	21 421C		LXI H,1C42
1C0A	36 11		MVI M,11
1C0C	7B	Q1:	MOV A,E
1C0D	17		RAL
1C0E	5F		MOV E,A
1C0F	7A		MOV A,D
1C10	17		RAL
1C11	57		MOV D,A
1C12	21 421C		LXI H,1C42
1C15	35		DCR M
1C16	E1		POP H
1C17	CA 311C		JZ Q2
1C1A	3E 00		MVI A,00
1C1C	17		RAL
1C1D	29		DAD H
1C1E	44		MOV B,H
1C1F	85		ADD L
1C20	2A 401C		LHLD 1C40
1C23	95		SUB L
1C24	4F		MOV C,A
1C25	78		MOV A,B
1C26	9C		SBB H
1C27	47		MOV B,A
1C28	C5		PUSH B
1C29	D2 0C1C		JNC Q1
1C2C	09		DAD B
1C2D	E3		XTHL
1C2E	C3 0C1C		JMP Q1
1C31	7A	Q2:	MOV A,D
1C32	2F		CMA
1C33	57		MOV D,A
1C34	7B		MOV A,E
1C35	2F		CMA
1C36	5F		MOV E,A
1C37	C9		RET

Unterprogramm DIV; es werden vier Hilfszellen benutzt: 1C40 . . . 1C43

Zusatz zu Übung 9: Prüfprogramm zur Geschwindigkeitsmessung

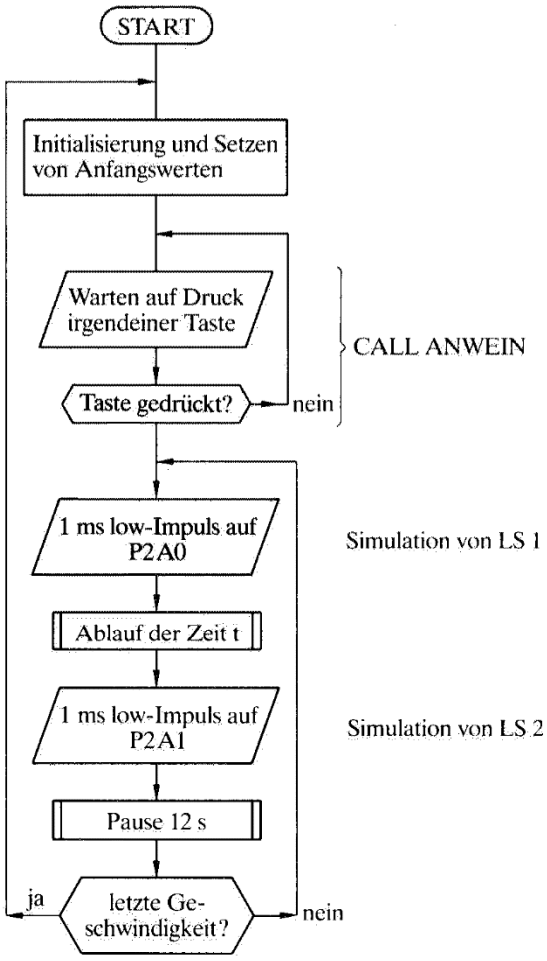
Wenn Schaltung und Programm wie oben gezeigt in Betrieb genommen wurden, dann empfiehlt es sich, zu prüfen, ob die errechneten und angezeigten Geschwindigkeits-Werte korrekt sind. Dazu müssen die beiden Lichtschranken-Signale simuliert werden, wobei ihr zeitlicher Abstand t genau bekannt sein muß. Falls Sie einen zweiten Experimentiercomputer ECB 85 zur Verfügung haben, kann er hierfür verwendet werden. Im folgenden wird nun das Prüfprogramm für diesen zweiten ECB 85 entwickelt.
Für die beiden Signale LS1 und LS2 werden die Port-Bits P2A0 und P2A1 des zweiten ECB 85 verwendet. Ihr Verlauf ist in der Skizze dargestellt:



Diese beiden Anschlüsse P2A0 und P2A1 des zweiten ECB 85 werden (wie LS1 und LS2, s. o.) auf die Port-Anschlüsse P2B0 und P2B1 des ersten ECB 85 gelegt. Dabei müssen die Masse-Anschlüsse der beiden Computer-Stromversorgungen miteinander verbunden werden.
Die Dauer der low-Impulse soll ca. 1 ms betragen, und ihr zeitlicher Abstand (t) voneinander hängt natürlich von der Geschwindigkeit ab, die simuliert werden soll (z. B.: $t = 65$ ms bei 11,08 km/h). Das zu entwickelnde Programm soll insgesamt 6 Geschwindigkeits-Werte simulieren, mit einer Pause von jeweils ca. 12 Sekunden nach jedem neuen Wert.
Das Flußdiagramm zeigt den Programmablauf. Das Warten auf die Betätigung irgendeiner Taste für den Beginn des Tests wird durch das Monitor-Unterprogramm ANWEIN realisiert, das ab Adresse 07D1 im Monitor-EPROM steht. Es ist also einfach der Befehl CALL ANWEIN (hex.: CD D1 07) auszuführen. Der Ablauf der Zeitspannen für die Geschwindigkeits-Simulation und für die 12-s-Pause ist durch ein einziges Unterprogramm ZEIT zu realisieren. Dies enthält eine Zeitschleife von 1 ms Dauer (vgl. *Übung 6*), die durch Herunterzählen des Registers B 5mal abläuft. Diese 5 ms laufen dann so oft ab, wie der Inhalt eines weiteren Registers (z. B. D) angibt. Dies wird beim Aufruf von ZEIT mit einer Zahl M geladen. Die verschiedenen Werte für M für die 6 vorzusehenden Geschwindigkeits-Werte und die dazwischenliegenden 5 Pausen werden in einem eigenen Speicherbereich hintereinander abgelegt und dann im Unterprogramm ZEIT durch indirekte Adressierung in das Register D geladen.

Zur Berechnung von M:

Zunächst ist gemäß dem Flußdiagramm das Hauptprogramm zu erstellen; siehe Programmliste. Aus ihr geht hervor, daß die gesamte Zeitspanne, die



Flußdiagramm

zwischen dem *Beginn* der beiden low-Impulse an P2A0 und P2A1 vergeht, folgende Dauer hat:
$t = (3146 + M * 15438) * 325$ ns.
Die 3146 Taktzyklen (jeder dauert 325 ns, siehe *Übung 6*) rühren von der 1-ms-Zeitschleife für P2A0 her sowie von den Befehlen im Adreßbereich 186D bis 1877, wobei auch der RET-Befehl im Unterprogramm ZEIT dazukommt. Durch den Wert M wird – wie gesagt – die Zeitspanne t festgelegt, die mit dem zu simulierenden Geschwindigkeits-Wert v wie folgt zusammenhängt:

$$t = \frac{720}{v\ (\text{km/h})}\ (\text{ms}) = \frac{720 \cdot 10^6}{v\ (\text{km/h})}\ (\text{ns})$$

Aus der für den gewählten Wert von v benötigten Zeit t wird dann gemäß der obigen Gleichung die Zahl M ermittelt. Für das Prüfprogramm sollen dabei auch zwei Geschwindigkeiten gewählt werden, die außerhalb des erlaubten Bereichs von 1 bis 99,99 km/h liegen. Da M eine ganze Zahl sein muß, ist nach der Festlegung von M dann die Geschwindigkeit v genau zu errechnen. Die nachfolgende Tabelle enthält 6 Beispielswerte.

M	t (ms)	v (km/h)
1	6.04	119.20
2	11.06	65.10
3	16.07	44.80
10 (0A H)	51.21	14.06
100 (64 H)	503.50	1.43
200 (C8 H)	1000.00	0.72

Wie schon erwähnt, werden die Werte von M im Speicher ab Adresse 18E0 hintereinander abgelegt. Dabei folgt auf jeden Wert für eine Testgeschwin-

Programm zur Prüfung der Geschwindigkeitsmessung:

Adresse	Hex.-Code	Labels	Mnemo-Code	Kommentar
1850	1E	START:	MVI E,06	Zähler laden
1	06			
2	3E		MVI A,01	P2A = Ausgabeport
3	01			
4	D3		OUT F0	
5	F0			
6	3E		MVI A,FF	P2A0 = 1
7	FF			P2A1 = 1
8	D3		OUT F1	
9	F1			
A	31		LXI SP,1C00	Stackpointer laden
B	00			
C	1C			
D	CD		CALL ANWEIN	Warten auf Druck irgendeiner Taste
E	D1			
F	07			
1860	21		LXI H,18E0	Adresse, unter der M abgelegt ist, nach HL
1	E0			
2	18			
3	3E	NEU:	MVI A,FE	
4	FE			
5	D3		OUT F1	
6	F1			
7	0E		MVI C,DB	P2A0 = 0 für 1 ms
8	DB			

Adresse	Hex.-Code	Labels	Mnemo-Code	Kommentar
9	0D	W2:	DCR C	
A	C2		JNZ W2	
B	69			
C	18			
D	3E		MVI A,FF	
E	FF			
F	D3		OUT F1	Ablauf von $t - 1$ ms (vgl. Text)
70	F1			
1	CD		CALL ZEIT	
2	B0			
3	18			
4	3E		MVI A,FD	
5	FD			
6	D3		OUT F1	
7	F1			
8	0E		MVI C,DB	
9	DB			P2A1 = 0 für 1 ms
A	0D	W3:	DCR C	
B	C2		JNZ W3	
C	7A			
D	18			
187E	3E		MVI A,FF	
F	FF			P2A0 = 1
1880	D3		OUT F1	P2A1 = 1
1	F1			
2	23		INX H	
3	3E		MVI A,FF	
4	FF			Adr. für M erhöhen; B ändern für Pause
5	32		STA 18B2	
6	B2			
7	18			
8	CD		CALL ZEIT	
9	B0			Pause
A	18			
B	23		INX H	
C	3E		MVI A,05	Adr. für M erhöhen
D	05			B wieder auf 5 setzen
E	32		STA 18B2	
F	B2			
1890	18			
1	1D		DCR E	Zähler dekrementieren
2	C2		JNZ NEU	
3	63			
4	18			
5	C3		JMP START	
6	50			
7	18			

digkeit laut obiger Tabelle ein anderer Wert, nämlich 2F H = 47 D für die Pause von ca. 12 Sekunden. Zusätzlich wird vor Beginn jeder Pause das Register B mit FF H = 255 D geladen (Befehle MVI A,FF und STA 18B2, siehe Programm). Nach Ablauf der Pause wird B wieder mit 5 belegt, damit die nächste Test-Zeitspanne nach Rücksprung zum Label NEU ablaufen kann.

Adresse	Hex.-Code	Labels	Mnemo-Code	Kommentar
18B0	56	ZEIT:	MOV D,M	
1	06	W1:	MVI B,05	
2	05			
3	0E	W:	MVI C,DB	
4	DB			
5	0D	AB:	DCR C	
6	C2		JNZ AB	
7	B5			
8	18			
9	05		DCR B	
A	C2		JNZ W	
B	B3			
C	18			
D	15		DCR D	
E	C2		JNZ W1	
F	B1			
C0	18			
1	C9		RET	

Unterprogramm ZEIT (M × B ms)

Adresse	Hex.-Code	M (dezimal)	
18E0	01	1	1 Geschw.
1	2F	47	Pause
2	02	2	2. Geschw.
3	2F	47	Pause
4	03	3	3. Geschw.
5	2F	47	Pause
6	0A	10	4. Geschw.
7	2F	47	Pause
8	64	100	5. Geschw.
9	2F	47	Pause
A	C8	200	6. Geschw.
B	2F	47	Pause

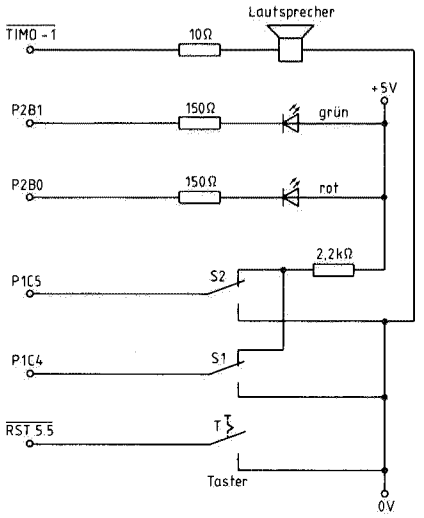
Datenteil mit den Werten für M (vgl. Text)

Übung 10: Reaktionszeit-Messung

In dieser Übung wird zum ersten Mal die Interrupt-Technik angewendet; d. h. das zu erwartende Signal von einer Taste wird nicht auf ein im Programm immer wieder abzufragendes Port-Bit eines 8155-Bausteins gegeben wie in den vorigen Übungen, sondern auf den Unterbrechungs-Eingang RST5.5 des Mikroprozessors. Wie aus *Bild 3.4.1* zu ersehen ist, muß an der Anwender-Schnittstelle das invertierte Signal $\overline{\text{RST5.5}}$ eingegeben werden, da im ECB 85 zwischen dem Anschluß a3 der Anwenderleiste und dem Mikroprozessor ein invertierendes Leistungs-Gatter geschaltet ist.

Aufgabe:

Es ist eine Anwender-Schaltung zu erstellen und dazu ein Programm auf dem ECB 85 zu entwickeln, mit denen folgendes realisiert werden soll: Abhängig von der Stellung eines Schalters S1 soll zum Start des Programms eine grüne Leuchtdiode leuchten; nach Ablauf einer Zufallszeit soll diese erlöschen und eine rote Diode aufleuchten. Danach soll der Benutzer dann möglichst schnell den Taster T betätigen (der mit $\overline{\text{RST5.5}}$ verbunden ist). Die vom Aufleuchten der roten Diode bis zum Betätigen von T verstrichene Zeitspanne ist auf einer vierstelligen Anzeige dezimal in Millisekunden auszugeben (vgl. hierzu auch *Übung 9*). Zusätzlich ist eine Fehlermeldung für den Fall vorzusehen, daß der Benutzer den Taster zu früh betätigt (vor dem Aufleuchten der roten Diode). Schließlich soll mit Hilfe eines weiteren Schalters S2 noch zu wählen sein zwischen der roten Diode und einem Lautsprecher als Signalgeber, der den Beginn der Reaktionszeit angibt.



Anwender-Schaltung zur Reaktionszeit-Messung. Widerstände jeweils ¼ W. Zur Verbindung der Anschlüsse mit der Anwenderleiste des ECB 85 siehe *Bild 3.4.1;* (+5 V und 0 V nicht vergessen!).

Die benötigten Anschlüsse am ECB 85

Wie aus dem Schaltbild hervorgeht, werden zunächst vier Port-Bits benötigt: zwei (P2B0 und P2B1) zur Ausgabe der Leuchtdioden-Zustände und zwei (P1C4 und P1C5) zum Einlesen der Schalter-Zustände S1 und S2. Weiterhin sind das Signal $\overline{\text{TIMER OUT}}$ des Bausteins 8155 Nr. 1 und der Anschluß $\overline{\text{RST5.5}}$ wie gezeigt zu verbinden.
Für die Ausgabe der Reaktionszeit werden – wie schon in *Übung 9* – die Ports A und B des 8155 Nr. 1 verwendet.

Initialisierung und Funktionstest der Hardware

Mit Kommando 4 ist auszugeben:
F0 = 02 (d. h. 02 auf Adresse F0: P2B ist Ausgabe-Kanal), und
F8 = 03 (d. h. 03 auf Adresse F8: P1A und P1B sind Ausgabe-Kanäle, P1C ist Eingabe-Kanal).
Nach dieser Initialisierung müssen die vier Anzeige-Ziffern jeweils Null zeigen, und die beiden Leuchtdioden müssen hell sein, da sie mit einem 0-Wert hellgesteuert werden (siehe Schaltbild).

Funktionsprüfung der Dioden

Die rote Diode liegt an Bit 0 des Ports 2B, und die grüne an Bit 1. Also können mit Kommando 4 die folgenden Zustände erzeugt werden:
F2 = 00 beide Dioden hell
F2 = 01 nur grüne Diode hell
F2 = 02 nur rote Diode hell
F2 = 03 beide Dioden dunkel.

Funktionsprüfung der Schalter

Hierzu wird mit Kommando 5 der Wert von Port C des 8155 Nr. 1 eingelesen; dieser Port hat die Adresse FB, so daß sich folgendes ergibt:
FB = xxx1 xxxx, wenn am Schalter S1 eine 1 anliegt (Port-Bit mit +5 V verbunden, wie gezeichnet)
FB = xxx0 xxxx, wenn am Schalter S1 eine 0 anliegt.

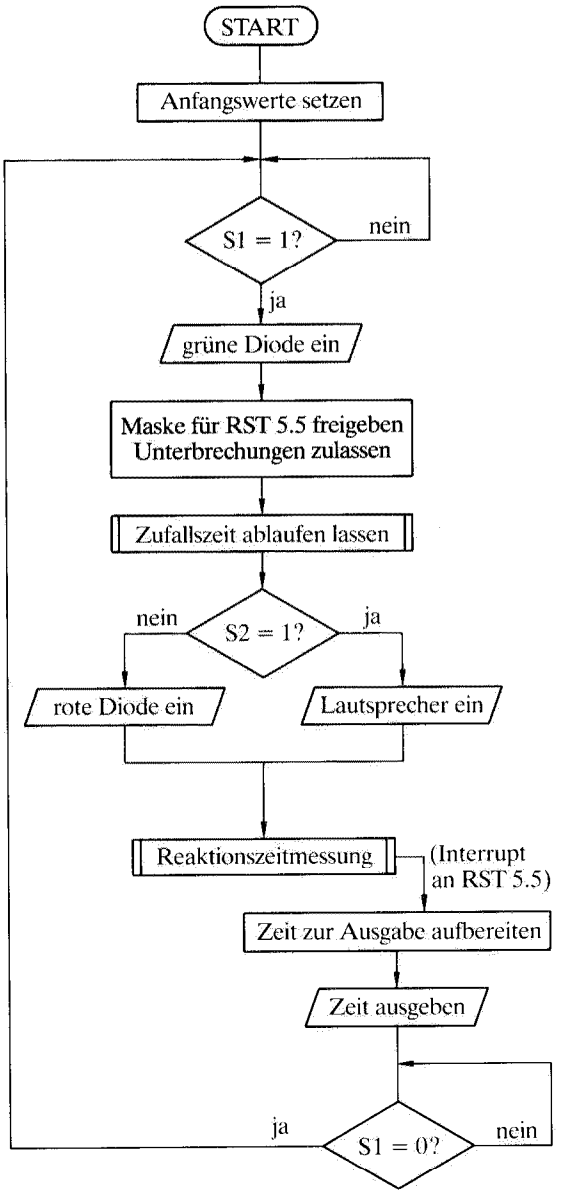
Entsprechend gilt für Schalter S2:
FB = xx1x xxxx, wenn eine 1 an S2 anliegt;
FB = xx0x xxxx, wenn eine 0 an S2 anliegt.

Funktionsprüfung des Lautsprechers

Mit Kommando 4 wird zunächst der Timer des 8155 Nr. 1 geladen:
FC = FF (Timer LSB) und
FD = 60 (Timer MSB, binär: 0110 0000).
Die Betriebsart-Bits des Timers sind die beiden höchstwertigen Bits im MSB, also hier 01, so daß gemäß *Bild 3.3.8* die Betriebsart 1 resultiert: Rechtecksignal. Wird nun der Timer gestartet mit
F8 = C3 (binär: 1100 0011),
so muß ein Ton erklingen. Mit dem auf die Adresse F8 ausgegebenen Wert C3 wird die oben beschriebene Initialisierung der E/A-Ports des 8155 Nr. 1 unverändert übernommen.

Anleitung zur Programm-Erstellung

Damit das Programm Unterbrechungs-Anforderungen bearbeiten kann, müssen bestimmte Vorbereitungen getroffen werden. Wenn, wie hier gewünscht, eine Unterbrechung über den Eingang RST5.5 des 8085 bearbeitet werden soll, dann muß die Unterbrechungsbedienungs-Routine bei der Adresse 002C beginnen. Denn beim RST5.5 wird auf die Adresse 8 * 5,5 = 44 gesprungen, und es ist 44 D = 2C H.
Im Monitor-Programm des ECB 85 steht in der Adresse 002C ein Sprungbefehl zur Adresse 082C; diese liegt im Adreßbereich des Anwender-EPROMs. Es muß also in den Anwender-Steckplatz ein EPROM eingesetzt werden, das in der Adresse 082C einen Sprungbefehl zu derjenigen Adresse im Programm enthält, bei der die Bearbeitung der Unterbrechung beginnt. In unserem Anwendungsfall sei dies die Adresse 18F0.
Ferner ist im Hauptprogramm die Unterbrechung mit Hilfe des Befehls EI (Enable Interrupt) zu ermöglichen und zuvor die Maske für den RST5.5 mit dem Befehl SIM (Set Interrupt Mask) freizugeben. Vor dem SIM-Befehl ist der Akku dazu mit 0E zu laden. Nach der Annahme der Unterbrechung muß sie wieder neu freigegeben werden, wenn eine weitere Messung erwünscht ist. Ähnlich wie in *Übung 9* liegt auch hier der Zahlenwert für die gemessene Zeitspanne als Dualzahl in einem Registerpaar vor und muß in eine 4stellige Dezimalzahl umgewandelt werden. Dies soll hier im Prinzip so wie in *Übung 9* durchgeführt werden. Das dazu aufzurufende Divisions-Unterprogramm DIV ist dasselbe wie in *Übung 9* (ab Adresse 1C00).
Das Flußdiagramm zeigt den Ablauf des zu erstellenden Programms im groben.



Flußdiagramm zur Reaktionszeitmessung.

Zur Lösung:

In der nachfolgend gezeigten Muster-Lösung ist keine echte Unterbrechungsbedienungs-Routine verwendet, sondern (wie oben schon erwähnt) ein einfacher Sprungbefehl zur Adresse 082C (und von dort nach 18F0). Es soll hier vor allem demonstriert werden, wie durch ein 1-Signal am Eingang RST5.5 des 8085 eine Programmverzweigung realisiert werden kann.

In der Praxis wird man aber den Prozessor viel besser ausnutzen, nämlich nach Bedienung der Unterbrechung wieder an die Aufrufstelle zurückkehren. Dabei ist die Unterbrechungsbedienungs-Routine so kurz wie möglich zu programmieren, damit das Hauptprogramm ohne große Störung ablaufen kann.

Im weiter unten wiedergegebenen zweiten Lösungs-Vorschlag ist dies so realisiert, daß im Unterprogramm RST55 nur abgefragt wird, ob zu früh reagiert wurde, und es wird der Wert FF in Adresse 1972 abgelegt als Kennzeichen dafür, daß die Unterbrechung erfolgte. Darauf wird im Hauptprogramm abgefragt (siehe ab Adresse 18B7).

Zum Ablauf des Programms (erster Lösungs-Vorschlag):

Das Programm wird nach der Eingabe (Anwender-EPROM mit Sprungbefehl nach 18F0 nicht vergessen!) mit Kommando 7 gestartet; S1 soll dann auf 1 stehen. Nach dem Aufleuchten der roten Diode ist zu reagieren (Taster T betätigen), und nach Ausgabe der Reaktionszeit auf der vierstelligen Anzeige (vgl. *Übung 9*) kann durch Umschalten von S1 auf 0 an den Programmbeginn verzweigt und mit Umschalten auf S1 = 1 eine neue Messung begonnen werden.

Programm zur Reaktionszeitmessung (1. Lösungsvorschlag):

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
1850	31		LXI SP,1C00	
1	00			
2	1C			
3	3E		MVI A,02	
4	02			
5	D3		OUT F0	
6	F0			Initialisierung
7	3E		MVI A,03	
8	03			
9	D3		OUT F8	
A	F8			
B	3E		MVI A,FF	
C	FF			
D	D3		OUT FC	
E	FC			Timerbytes vorbelegen, so daß Tonausgabe möglich
F	3E		MVI A,60	
60	60			
1	D3		OUT FD	
2	FD			
3	3E	ARB:	MVI A,FF	Dioden löschen
4	FF			
5	D3		OUT F2	
6	F2			
7	DB	ABF:	IN FB	
8	FB			S1 abfragen
9	E6		ANI 10	

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
A	10			
B	CA		JZ ABF	
C	67			
D	18			
E	3E		MVI A,01	grüne Diode ein
F	01			
70	D3		OUT F2	
1	F2			
2	3E		MVI A,0E	Maske für
3	0E			RST 5.5
4	30		SIM	freigeben
5	FB		EI	Unterbrechung zulass.
6	CD		CALL ZEIT	Zufallszeit
7	D0			ablaufen lassen
8	18			
9	01		LXI B,0000	B,C mit Null laden
A	00			(für späteres Zählen)
B	00			
C	DB		IN FB	
D	FB			
E	00		NOP	S2 abfragen; wenn
F	00		NOP	S2 = 0, dann rote
80	E6		ANI 20	Diode ein
1	20			
2	CA		JZ DIODE	
3	8E			
4	18			
5	3E		MVI A,C3	sonst:
6	C3			Lautsprecher ein
7	D3		OUT F8	
8	F8			
1889	1E	W:	MVI E,01	
A	01			Softwareschalter
B	C3		JMP MISS	setzen und mit der
C	B0			Messung beginnen
D	18			
E	3E	DIODE:	MVI A,02	
F	02			
90	D3		OUT F2	rote Diode ein
1	F2			
2	C3		JMP W	
3	89			
4	18			
18B0	CD	MISS:	CALL UP	
1	B7			Reg.-Paar BC im
2	18			Millisekunden-Takt
3	03		INX B	hochzählen, bis
4	C3		JMP MISS	RST 5.5 kommt

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
5	B0			
6	18			
7	3E	UP:	MVI A,01	
8	01			
9	26	UP 1:	MVI H,D6	
A	D6			
B	25	AB:	DCR H	
C	C2		JNZ AB	
D	BB			1 ms Zeiteinheit
E	18			
F	3D		DCR A	
C0	C2		JNZ UP 1	
1	B9			
2	18			
3	C9		RET	
18D0	06	ZEIT:	MVI B,5F	
1	5F			
2	16	AB2:	MVI D,0F	
3	0F			
4	0E	AB1:	MVI C,FF	
5	FF			
6	0D	AB3:	DCR C	
7	C2		JNZ AB3	
8	D6			ab dem 2. Programm-
9	18			lauf: Zufallszeit
A	15		DCR D	(vgl. Text)
B	C2		JNZ AB1	
C	D4			
D	18			
E	05		DCR B	
F	C2		JNZ AB2	
E0	D2			
1	18			
2	C9		RET	
18F0	7B	RST 55:	MOV A,E	
1	C6		ADI 0	RST ist angenommen
2	00			worden, der Software-
3	CA		JZ PFUI	Schalter E wird über-
4	50			prüft; ist er *Null,*
5	19			*dann* Ausgabe: PFUI,
6	3E		MVI A,43	*sonst:* Timer-Stop,
7	43			Dioden löschen
8	D3		OUT F8	
9	F8			
A	3E		MVI A,03	
B	03			

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
C	D3		OUT F2	
D	F2			
E	50		MOV D,B	
F	59		MOV E,C	
1900	21		LXI H,000A	
1	0A			
2	00			
3	CD		CALL DIV	
4	00			
5	1C			
6	7D		MOV A,L	
7	32		STA HILF	
8	70			
9	19			
A	21		LXI H,000A	
B	0A			
C	00			
D	CD		CALL DIV	
E	00			
F	1C			
10	7D		MOV A,L	
1	32		STA HILF+1	
2	71			
3	19			Aufbereitung und
4	21		LXI H,000A	Ausgabe der
5	0A			Reaktionszeit
6	00			
7	CD		CALL DIV	
8	00			
9	1C			
A	7B		MOV A,E	
B	07		RLC	
C	07		RLC	
D	07		RLC	
E	07		RLC	
F	B5		ORA L	
20	D3		OUT F9	
1	F9			
2	3A		LDA HILF+1	
3	71			
4	19			
5	07		RLC	
6	07		RLC	
7	07		RLC	
8	07		RLC	
1929	21		LXI H,HILF	
A	70			
B	19			

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
C	B6		ORA M	
D	D3		OUT FA	
E	FA			
F	79		MOV A,C	Wert in C
30	32		STA ZEIT+1	für die neue
1	D1			Zufallszeit in
2	18			UP ZEIT ablegen
3	DB	WART:	IN FB	
4	FB			
5	E6		ANI 10	S1-Abfrage
6	10			
7	C2		JNZ WART	
8	33			
9	19			
A	1E		MVI E,0	
B	00			Neue Messung
C	31		LXI SP,1C00	vorbereiten
D	00			
E	1C			
F	3E		MVI A,C1	
40	C1			ECB-Anzeige
1	D3		OUT EF	löschen
2	EF			
3	DB	WA:	IN EF	
4	EF			
5	E6		ANI 80	
6	80			
7	C2		JNZ WA	Warten, bis Anzeige
8	43			verfügbar
9	19			
A	C3		JMP ARB	
B	63			
C	18			
1950	00	PFUI:	NOP	
1	CD		CALL ANWAZL	
2	BC			
3	07			
4	3E		MVI A,92	
5	92			
6	D3		OUT EF	
7	EF			
8	3E		MVI A,73	
9	73			
A	D3		OUT EE	Ausgabe von PFUI
B	EE			wenn zu früh
C	3E		MVI A,71	„reagiert" wurde
D	71			

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
E	D3		OUT EE	
F	EE			
60	3E		MVI A,3E	
1	3E			
2	D3		OUT EE	
3	EE			
4	3E		MVI A,06	
5	06			
6	D3		OUT EE	
7	EE			
8	C3		JMP WART	
9	33			
A	19			
082C	C3		JMP RST55	
D	F0			Sprungbefehl im
E	18			Anwender-EPROM
1970		HILF:		(Hilfszellen zum
1971				Zwischenspeichern)

DIV ist das bekannte Divisionsprogramm aus *Übung 9*, abgelegt ab Adresse 1C00:

DIVIDEND in D u. E
DIVISOR in H u. L
QUOTIENT in D u. E
REST in H u. L

Zum Erzeugen der Zufallszeit:

Im Unterprogramm ZEIT wird die Zeitspanne bis zum Aufleuchten der roten Diode bzw. bis zum Ertönen des Lautsprechers erzeugt. Beim *ersten* Programmlauf (z. B. zum Test) ist dies eine feststehende Zeitspanne, da ab Adresse 18D0 die Register B, C und D jeweils mit einem bestimmten Wert geladen werden. In diesem ersten (und in jedem weiteren) Programmlauf wird aber nach erfolgtem Interrupt zur Aufbereitung der Reaktionszeit das Unterprogramm DIV aufgerufen, in dem das Register C verwendet, sein Inhalt also verändert wird. Nach dem Verlassen des Divisions-Unterprogramms kann somit der Inhalt des Registers C abhängig von der Reaktionszeit als zufällig betrachtet werden. Nach Ausgabe des Ergebnisses auf der ECB-Anzeige wird bei Adresse 192F und 1930 der Inhalt von C mit den Befehlen MOV A,C und STA ZEIT+1 (bzw. STA 18D1) auf der Adresse 18D1 abgespeichert; der erste Befehl des Unterprogramms ZEIT lautet damit dann für den nächsten Programmlauf: MVI B,XX, wobei XX der oben erwähnte zufällige Wert in C (im vorhergehenden Programmlauf) ist.

Zur Ausgabe der Fehlermeldung "PFUI" bei zu früher „Reaktion":

Zu Beginn dieses Programmteils (ab Adresse 1950) wird mit Aufruf des Monitor-Unterprogramms ANWAZL (vgl. hierzu *Übung 11*) die ECB-Anzeige gelöscht und dann das Kommandowort 92 an den Anzeigesteuer-Baustein 8279 ausgegeben (Adresse EF, vgl. *Bild 3.3.5*). Anschließend werden die vier 7-Segment-Muster (vgl. hierzu auch *Übung 7*) auf die Anzeigestellen des ECB 85 ausgegeben. Die Ausgabe-Adresse ist hier jeweils EE.

Programm zur Reaktionszeitmessung (2. Lösungsvorschlag):

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
1850	31	NEU:	LXI SP,1C00	
1	00			
2	1C			
3	3E		MVI A,02	
4	02			
5	D3		OUT F0	
6	F0			
7	3E		MVI A,03	
8	03			Initialisierung
9	D3		OUT F8	
A	F8			
B	3E		MVI A,FF	
C	FF			
D	D3		OUT FC	
E	FC			
F	3E		MVI A,60	
60	60			
1	D3		OUT FD	
2	FD			
3	3E	ARB:	MVI A,FF	Dioden löschen
4	FF			
5	D3		OUT F2	
6	F2			
7	DB	ABF:	IN FB	
8	FB			S1 abfragen
9	E6		ANI 10	
A	10			
B	CA		JZ ABF	
C	67			
D	18			
E	3E		MVI A,01	grüne Diode ein
F	01			
70	D3		OUT F2	
1	F2			
2	3E		MVI A,0E	Maske für
3	0E			RST 5.5
4	30		SIM	freigeben
5	FB		EI	Unterbrechung zulass.
6	CD		CALL ZEIT	Zufallszeit
7	D0			ablaufen lassen
8	18			
9	01		LXI B,0000	
A	00			
B	00			
C	DB		IN FB	
D	FB			
E	00		NOP	S2 abfragen

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
F	00		NOP	
80	E6		ANI 20	
1	20			
2	CA		JZ DIODE	
3	95			
4	18			
5	3E		MVI A,C3	Lautsprecher ein
6	C3			
7	D3		OUT F8	
8	F8			
1889	1E	W:	MVI E,01	Software-Schalter E setzen; Null auf INT geben
A	01			
B	3A		LDA INT	
C	72			
D	19			
E	97		SUB A	
F	32		STA INT	
90	72			
1	19			
2	C3		JMP MISS	
3	B0			
4	18			
5	3E	DIODE:	MVI A,02	rote Diode ein
6	02			
7	D3		OUT F2	
8	F2			
9	C3		JMP W	
A	89			
B	18			
18B0	CD	MISS:	CALL UP	Reaktionszeit-messung
1	BF			
2	18			
3	03		INX B	
4	3A		LDA INT	
5	72			
6	19			
7	C6		ADI 01	Abfrage, ob bereits unterbrochen wurde
8	01			
9	DA		JC RST 551	
A	F0			
B	18			
C	C3		JMP MISS	
D	B0			
E	18			
F	3E	UP:	MVI A,01	
C0	01			
1	26	UP1:	MVI H,D4	
2	D4			

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
3	25	AB:	DCR H	
4	C2		JNZ AB	auf 1ms korrigierte
5	C3			Gesamtschleife
6	18			
7	3D		DCR A	
8	C2		JNZ UP1	
9	C1			
A	18			
B	C9		RET	
18D0	06	ZEIT:	MVI B,5F	
1	5F			
2	16	AB2:	MVI D,0F	
3	0F			
4	0E	AB1:	MVI C,FF	
5	FF			
6	0D	AB3:	DCR C	
7	C2		JNZ AB3	
8	D6			ab dem 2. Programm-
9	18			lauf: Zufallszeit
A	15		DCR D	(vgl. Text)
B	C2		JNZ AB1	
C	D4			
D	18			
E	05		DCR B	
F	C2		JNZ AB2	
E0	D2			
1	18			
2	C9		RET	
18F0	7B	RST 551:	MOV A,E	
1	C6		ADI 00	
2	00			Abfrage, ob zu früh
3	CA		JZ PFUI	„reagiert" wurde
4	50			
5	19			
6	3E		MVI A,43	
7	43			
8	D3		OUT F8	
9	F8			
A	3E		MVI A,03	
B	03			
C	D3		OUT F2	
D	F2			
E	50		MOV D,B	
F	59		MOV E,C	
1900	21		LXI H,000A	
1	0A			
2	00			

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
3	CD		CALL DIV	
4	00			
5	1C			
6	7D		MOV A,L	
7	32		STA HILF	
8	70			
9	19			
A	21		LXI H,000A	
B	0A			
C	00			
D	CD		CALL DIV	Aufbereitung und
E	00			Ausgabe der
F	1C			Reaktionszeit
10	7D		MOV A,L	
1	32		STA HILF+1	
2	71			
3	19			
4	21		LXI H,000A	
5	0A			
6	00			
7	CD		CALL DIV	
8	00			
9	1C			
A	7B		MOV A,E	
B	07		RLC	
C	07		RLC	
D	07		RLC	
E	07		RLC	
F	B5		ORA L	
20	D3		OUT F9	
1	F9			
2	3A		LDA HILF+1	
3	71			
4	19			
5	07		RLC	
6	07		RLC	
7	07		RLC	
8	07		RLC	
1929	21		LXI H,HILF	
A	70			
B	19			
C	B6		ORA M	
D	D3		OUT FA	
E	FA			
F	79		MOV A,C	
30	32		STA ZEIT+1	C für die neue
1	D1			Zufallszeit auf Adr.
2	18			18D1 geben

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
3	DB	WART:	IN FB	
4	FB			
5	E6		ANI 10	S1-Abfrage
6	10			
7	C2		JNZ WART	
8	33			
9	19			
A	1E		MVI E,00	
B	00			Neue Messung
C	00		NOP	vorbereiten
D	00		NOP	
E	00		NOP	
F	3E		MVI A,C1	
40	C1			
1	D3		OUT EF	
2	EF			
3	06		MVI B,01	Prellzeit abwarten
4	01			
5	CD		CALL ZEIT+2	
6	D2			
7	18			
8	C3		JMP ARB	
9	63			
A	18			
1950	00	PFUI:	NOP	
1	CD		CALL ANWAZL	
2	BC			
3	07			
4	3E		MVI A,92	
5	92			
6	D3		OUT EF	
7	EF			
8	3E		MVI A,73	
9	73			
A	D3		OUT EE	
B	EE			
C	3E		MVI A,71	
D	71			
E	D3		OUT EE	Ausgabe von PFUI
F	EE			
60	3E		MVI A,3E	
1	3E			
2	D3		OUT EE	
3	EE			
4	3E		MVI A,06	
5	06			
6	D3		OUT EE	
7	EE			

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
8	C3		JMP NEU	
9	50			
A	18			
1980	F5	RST 55:	PUSH PSW	
1	7B		MOV A,E	
2	C6		ADI 00	
3	00			Abfrage, ob zu früh
4	CA		JZ FPUI	„reagiert" wurde
5	50			
6	19			
7	3E		MVI A,FF	
8	FF			
9	32		STA INT	
A	72			
B	19			
C	F1		POP PSW	
D	C9		RET	
1970		HILF:		Hilfszellen zur Ausgabe
1				
2		INT:		Softwareschalter für Unterbrechung
082C	C3		JMP RST55	(im EPROM andere Sprungadresse als beim 1. Lösungsvorschlag)
D	80			
E	19			

Übung 11: Der ECB 85 als Uhr

In dieser Übung soll der ECB 85 als Uhr programmiert werden, wobei folgende Bedingungen zu erfüllen sind:
1. Der nicht verbietbare Unterbrechungs-Eingang TRAP des 8085 (bzw. $\overline{\text{TRAP}}$ an der Anwender-Schnittstelle des ECB 85) ist für eine Unterbrechungsbedienungs-Routine zu verwenden.
2. Die beiden Timer des ECB 85 in den Bausteinen 8155 sind für die Erzeugung des Zeittaktes zu benutzen.
3. Die Anzeige des ECB 85 soll die Zeit gemäß dem Muster hh=mm=ss anzeigen; dabei sind hh die Stunden, mm die Minuten und ss die Sekunden.
Diese Übung dient vor allem dazu, einige Monitor-Unterprogramme des ECB 85 für die Tastatur-Eingabe und die Anzeige kennenzulernen.

Anleitungen zur Lösung:

Es sind folgende *Hardware-Maßnahmen* durchzuführen (zur Anschlußbelegung an der Anwender-Schnittstelle des ECB 85 siehe *Bild 3.4.1*):
$\overline{\text{TIMO-1}}$ mit TIMI-2 verbinden; die beiden Timer sind dann „hintereinandergeschaltet".
$\overline{\text{TIMO-2}}$ mit $\overline{\text{TRAP}}$ verbinden; ein low-Impuls an $\overline{\text{TIMO-2}}$ bewirkt dann eine Unterbrechungs-Anforderung.
Software-Maßnahmen: Die Timer sind in Betriebsart 3 (vgl. *Bild 3.3.8 und 3.3.9*) zu programmieren; da sie hintereinandergeschaltet sind, ergibt sich für eine Grundzeit von 1 Sekunde:
$x_1 \cdot 325\ \text{ns} \cdot x_2 = 1\ \text{s}$,
wobei Timer 1 mit x_1 und Timer 2 mit x_2 zu laden ist.
Zum Stellen der Uhr soll die Tastatur des ECB 85 benutzt werden, wobei dies mit Hilfe des Monitor Unterprogramms ANWEIN erfolgen soll. Zur Anzeige der Uhrzeit werden folgende Monitor-Unterprogramme benötigt: ANWAZL, UMW und ANWAUS.

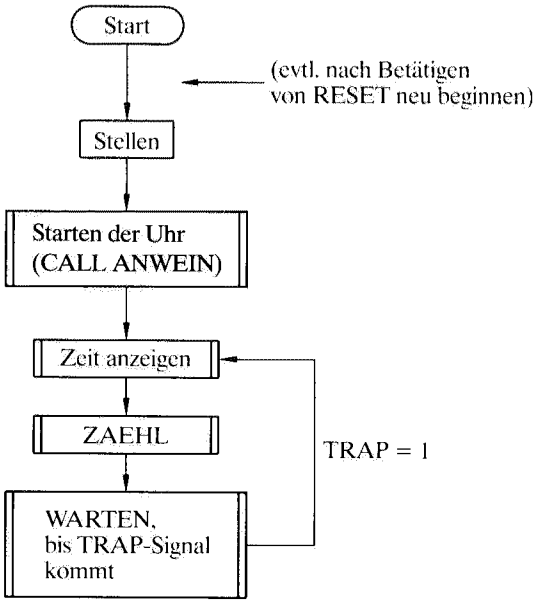
Kurze Beschreibung der Monitor-Unterprogramme (das Listing finden Sie im Anhang):

ANWEIN (ab Adresse 07D1):
Der Pufferspeicher für die Tastatur-Eingabe wird gelöscht, und es wird gewartet, bis eine Tastatur-Eingabe erfolgt. Wird ein Zeichen eingegeben, dann erfolgt der Rücksprung mit Übergabe des Zeichen-Codes im Akku.

ANWAZL (ab Adresse 07BC):
Die Anzeige des ECB 85 wird gelöscht; alle 8 Anzeigestellen sind dunkel.
UMW (ab Adresse 05C9):
Das niederwertige Halbbyte des Akku-Inhalts wird in ein 7-Segment-Muster umgewandelt; dies steht anschließend ebenfalls im Akku.
ANWAUS (ab Adresse 07C7):
Der im Akku befindliche 7-Segment-Code wird an die Anzeige des ECB 85 ausgegeben. Die Nr. der Anzeigestelle muß im Register B stehen (0 bis 7, von links nach rechts gezählt).

Behandlung der Unterbrechung mittels TRAP-Signal:

Bei Annahme der Unterbrechung springt das Monitor-Programm zur Anwender-EPROM-Adresse 0824. Hier muß also ein Sprung zur Unterbrechungs-Behandlung im Hauptprogramm stehen.



Flußdiagramm

Lösung

Die beiden Timer-Bytes sind wie folgt zu laden (vgl. *Bild 3.3.8* zur Betriebsart):
Timer 1 mit FA93 H, d. h. mit 14995 D und Betriebsart 3 (binär 11); dies ergibt jeweils nach 4873,375 ms einen low-Impuls von 325 ns Dauer.
Timer 2 mit C0CD H, d. h. mit 205 D und Betriebsart 3 (binär 11); dies ergibt jeweils nach 999 ms einen low-Impuls von 4 ms Dauer.
Hierbei ist berücksichtigt, daß im ECB 85 der Eingang des Timers 1 mit dem Signal CLK verbunden ist (Periodendauer: 325 ns).

Das *Flußdiagramm* zeigt den Ablauf des Programms im groben, und alle näheren Erläuterungen sind der Spalte „Bemerkungen" des Programm-Listings zu entnehmen.
Das Programm wird zunächst zur Eingabe der aktuellen Uhrzeit (nächste volle Minute) mit Kommando 7 bei Adresse 1810 gestartet (Sprungadresse im Anwender-EPROM bei der Programm-Eingabe nicht vergessen!) und wartet dann bei Adresse 1883 auf das Betätigen der Quittungs-Taste, die genau bei der vollen Minute zu drücken ist.

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
1810	31 00 1C		LXI SP,1C00	SP laden
13	CD BC 07		CALL ANWAZL	Anzeige löschen
16	21 01 1B		LXI H,1B01	Stellenzähler laden
19	06 00		MVI B,00	1. Stelle Anzeige,
1B	CD D1 07		CALL ANWEIN	1. Stelle Stunden
1E	E6 0F		ANI 0F	einlesen
20	CA 2D 18		JZ A1	
23	FE 02		CPI 02	
25	CA 32 18		JZ A2	
28	1E 0A		MVI E,0A	wenn 1. Stelle = 1:
2A	C3 34 18		JMP A3	10 → E
2D	1E 00	A1:	MVI E,00	wenn 1. Stelle = 0:
2F	C3 34 18		JMP A3	00 → E
32	1E 14	A2:	MVI E,14	wenn 1. Stelle = 2: 20 → E
34	77	A3:	MOV M,A	
35	CD C9 05		CALL UMW	„Tastenecho"
38	CD C7 07		CALL ANWAUS	
3B	04		INR B	Stellenzähler u. Anzeigestelle erhöhen
3C	2C		INR L	
3D	CD D1 07		CALL ANWEIN	} 2. Stelle Stunden einlesen
40	E6 0F		ANI 0F	
42	77		MOV M,A	
43	CD C9 05		CALL UMW	„Tastenecho"
46	CD C7 07		CALL ANWAUS	Stunden ausgeben (2mal)
49	7E		MOV A,M	2. Stelle der Stunden auf E aufaddieren
4A	83		ADD E	
4B	5F		MOV E,A	
4C	3E 18		MVI A,18	24 D → A
4E	93		SUB E	} E = 24D – E = Anzahl der Stunden bis 00 Uhr („Std.-Rest")
4F	5F		MOV E,A	
50	CD 9C 18		CALL ZEICH	„=" ausgeben
53	04	B1:	INR B	Anzeigestelle und
54	2C		INR L	Stellenzähler erhöhen

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
55	CD D1 07		CALL ANWEIN	Minuten
58	E6 0F		ANI 0F	einlesen
5A	77		MOV M,A	(geschieht für 2 Stellen)
185B	CD C9 05		CALL UMW	„Tastenecho"
5E	CD C7 07		CALL ANWAUS	Minuten ausgeben (2mal)
61	0D		DCR C	
62	C2 53 18		JNZ B1	
65	CD 9C 18		CALL ZEICH	„=" ausgeben
68	04	B2:	INR B	Anzeigestelle und
69	2C		INR L	Stellenzähler erhöhen
6A	3E 00		MVI A,00	
6C	77		MOV M,A	0 für Sekunden ausgeben
6D	CD C9 05		CALL UMW	(2 mal)
70	CD C7 07		CALL ANWAUS	
73	0D		DCR C	
74	C2 68 18		JNZ B2	
77	3E CD		MVI A,CD	
79	D3 F4		OUT F4	
7B	3E C0		MVI A,C0	Timer 2 mit 00CD und Betriebsart 3
7D	D3 F5		OUT F5	(binär: 11) laden und starten
7F	3E C1		MVI A,C1	
81	D3 F0		OUT F0	
83	CD D1 07	B3:	CALL ANWEIN	
86	E6 1F		ANI 1F	Start der Uhr, wenn
88	D6 13		SUI 13	Quittungstaste gedrückt wird
8A	C2 83 18		JNZ B3	
8D	3E 93		MVI A,93	
8F	D3 FC		OUT FC	
91	3E FA		MVI A,FA	Timer 1 mit 3A93 und Betriebsart 3
93	D3 FD		OUT FD	(binär 11) laden und starten
95	3E C1		MVI A,C1	
97	D3 F8		OUT F8	
99	CD AA 18		CALL ZAEHL	
189C	0E 02	ZEICH:	MVI C,02	
9E	04		INR B	
9F	3E 48		MVI A,48	7-Segment-Code für Zeichen =
A1	CD C7 07		CALL ANWAUS	
A4	C9		RET	
18A5	00	WARTEN:	NOP	Unterprogramm läuft,
A6	C3 A5 18		JMP WARTEN	bis TRAP-Signal kommt
A9	00		NOP	
18AA	2E 06	ZAEHL:	MVI L,06	Stellenzähler auf zweite
AC	7E		MOV A,M	Sekundenstelle. Wenn Einer der
AD	FE 09		CPI 09	Sekunden noch nicht 9, dann
AF	CA B6 18		JZ C1	um 1 erhöhen und WARTEN,
B2	34		INR M	sonst löschen
B3	C3 A5 18		JMP WARTEN	

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
55	CD D1 07		CALL ANWEIN	Minuten
58	E6 0F		ANI 0F	einlesen
5A	77		MOV M,A	(geschieht für 2 Stellen)
185B	CD C9 05		CALL UMW	„Tastenecho"
5E	CD C7 07		CALL ANWAUS	Minuten ausgeben (2mal)
61	0D		DCR C	
62	C2 53 18		JNZ B1	
65	CD 9C 18		CALL ZEICH	„=" ausgeben
68	04	B2:	INR B	Anzeigestelle und
69	2C		INR L	Stellenzähler erhöhen
6A	3E 00		MVI A,00	
6C	77		MOV M,A	0 für Sekunden ausgeben
6D	CD C9 05		CALL UMW	(2 mal)
70	CD C7 07		CALL ANWAUS	
73	0D		DCR C	
74	C2 68 18		JNZ B2	
77	3E CD		MVI A,CD	
79	D3 F4		OUT F4	
7B	3E C0		MVI A,C0	Timer 2 mit 00CD und Betriebsart 3
7D	D3 F5		OUT F5	(binär: 11) laden und starten
7F	3E C1		MVI A,C1	
81	D3 F0		OUT F0	
83	CD D1 07	B3:	CALL ANWEIN	
86	E6 1F		ANI 1F	Start der Uhr, wenn
88	D6 13		SUI 13	Quittungstaste gedrückt wird
8A	C2 83 18		JNZ B3	
8D	3E 93		MVI A,93	
8F	D3 FC		OUT FC	
91	3E FA		MVI A,FA	Timer 1 mit 3A93 und Betriebsart 3
93	D3 FD		OUT FD	(binär 11) laden und starten
95	3E C1		MVI A,C1	
97	D3 F8		OUT F8	
99	CD AA 18		CALL ZAEHL	
189C	0E 02	ZEICH:	MVI C,02	
9E	04		INR B	
9F	3E 48		MVI A,48	7-Segment-Code für Zeichen =
A1	CD C7 07		CALL ANWAUS	
A4	C9		RET	
18A5	00	WARTEN:	NOP	Unterprogramm läuft,
A6	C3 A5 18		JMP WARTEN	bis TRAP-Signal kommt
A9	00		NOP	
18AA	2E 06	ZAEHL:	MVI L,06	Stellenzähler auf zweite
AC	7E		MOV A,M	Sekundenstelle. Wenn Einer der
AD	FE 09		CPI 09	Sekunden noch nicht 9, dann
AF	CA B6 18		JZ C1	um 1 erhöhen und WARTEN,
B2	34		INR M	sonst löschen
B3	C3 A5 18		JMP WARTEN	

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
B6	36 00	C1:	MVI M,00	
B8	2D		DCR L	
B9	7E		MOV A,M	Stellenzähler auf 1. Sek.-Stelle,
BA	FE 05		CPI 05	wenn Zehner der Sek. noch nicht 5,
BC	CA C3 18		JZ C2	dann um 1 erhöhen und WARTEN,
BF	34		INR M	sonst löschen
C0	C3 A5 18		JMP WARTEN	
C3	36 00	C2:	MVI M,00	
C5	2D		DCR L	
C6	7E		MOV A,M	Stellenzähler auf 2. Min.-Stelle,
C7	FE 09		CPI 09	wenn Minuten-Einer noch nicht 9,
C8	CA D0 18		JZ C3	WARTEN und um 1 erhöhen,
CC	34		INR M	sonst löschen
CD	C3 A5 18		JMP WARTEN	
D0	36 00	C3:	MVI M,00	
D2	2D		DCR L	
D3	7E		MOV A,M	Stellenzähler auf 1. Min.-Stelle,
D4	FE 05		CPI 05	wenn Minuten-Zehner noch nicht 5
D6	CA DD 18		JZ C4	WARTEN und um 1 erhöhen;
D9	34		INR M	sonst löschen
DA	C3 A5 18		JMP WARTEN	
DD	36 00	C4:	MVI M,00	
DF	2D		DCR L	Stellenzähler auf 2. Std.-Stelle
E0	1D		DCR E	E = E – 1 (Std.-Rest wird heruntergezählt)
E1	CA F6 18		JZ C6	wenn Rest noch nicht 0, dann:
E4	7E		MOV A,M	Stunde in Akku
18E5	00		NOP	
E6	FE 09		CPI 09	wenn Stunde noch nicht 9,
E8	CA EF 18		JZ C5	dann um 1 erhöhen und
EB	34		INR M	WARTEN, sonst löschen
EC	C3 A5 18		JMP WARTEN	
EF	36 00	C5:	MVI M,00	
F1	2D		DCR L	Stellenzähler auf 1. Stunden-Stelle
F2	34		INR M	Std.-Wert erhöhen
F3	C3 A5 18		JMP WARTEN	WARTEN
F6	36 00	C6:	MVI M,00	Std.-Rest = 0 (vgl. Adr. 18E1)
F8	2D		DCR L	Stellenzähler auf 1. Std.-Stelle
F9	36 00		MVI M,00	0 → Stunden;
FB	1E 18		MVI E,18	E wieder mit 24 laden (Std.-Rest)
FD	C3 A5 18		JMP WARTEN	
190A	06 07	AUSG:	MVI B,07	hintere Anzeigestelle (ganz rechts)
0C	2E 06		MVI L,06	2. Sekundenstelle
0E	7E	D1:	MOV A,M	
0F	CD C9 05		CALL UMW	Sek.-Einer in Akku
12	CD C7 07		CALL ANWAUS	und auf Anzeige ausgeben
15	05		DCR B	Anzeigestelle und Stellenzähler
16	2D		DCR L	erniedrigen

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
17	7E		MOV A,M	Sek.-Zehner in Akku
18	CD C9 05		CALL UMW	und auf Anzeige ausgeben
1B	CD C7 07		CALL ANWAUS	
1E	05		DCR B	Stellenzähler auf Minuten-Einer
1F	05		DCR B	einstellen
20	2D		DCR L	
21	C2 0E 19		JNZ D1	bis L = 0: Ausgabe von
24	C3 AA 18		JMP ZAEHL	Minuten und Stunden
0824	33		INX SP	
25	33		INX SP	Nach Unterbrechung durch TRAP:
26	C3 0A 19		JMP 190A	Sprung zu AUSG

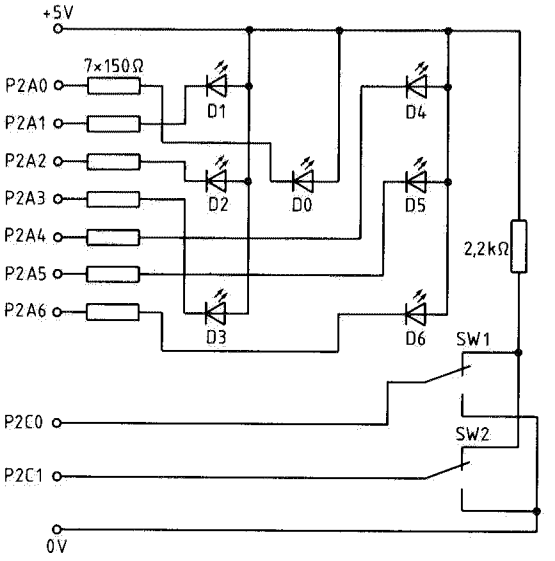
Übung 12: Ein Würfelprogramm

Mit Hilfe einer hierfür neu zu erstellenden Anwender-Schaltung soll der ECB 85 in dieser Übung als Würfel verwendet werden. Die dafür benötigten Unterprogramme des ECB-85-Monitors und die Port-Anschlüsse der 8155-Bausteine sind in den vorigen Übungen schon behandelt worden, so daß es hier jetzt mehr um die Festigung des bisher Gelernten geht.
Wie das weiter unten wiedergegebene Schaltbild zeigt, werden 7 Leuchtdioden so angeordnet, daß beim Aufleuchten von mindestens 1 und höchstens 6 der Dioden ein Muster entstehen kann, wie es ein normaler Würfel auch zeigt.

Aufgabe:

Es ist ein Programm für den ECB 85 zu schreiben, das die in der Abbildung gezeigte Dioden-Anordnung wie einen Würfel arbeiten läßt. Mit dem Schalter SW1 soll der Würfelvorgang einzuleiten bzw. zu stoppen sein. Der Schalter SW2 soll die Möglichkeit geben, über eine zunächst unbestimmte Anzahl von Würfen (höchstens 16) die Augensumme auf zwei 7-Segment-Anzeigen auszugeben (vgl. hierzu auch *Übung 9*). Die Anzahl der Würfelvorgänge ist auf der Anzeige des ECB 85 auszugeben (vgl. *Übung 10*).
Im einzelnen sollen die Schalterstellungen folgende Bedeutung haben (x = Schalterstellung ohne Bedeutung):

SW1	SW2	
0	0	es wird gewürfelt;
0	1	Würfelvorgang wird gestoppt, und die erwürfelte Augen-Kombination wird angezeigt;
1	x	ab jetzt werden alle erwürfelten Resultate addiert, und das Ergebnis wird ausgegeben (s. u.).



Anwenderschaltung für die Benutzung des ECB 85 als Würfel.

Beschreibung des zu realisierenden Programmablaufs (vgl. Flußdiagramm):

Beim Programmstart sollen SW1 und SW2 auf 0 stehen. Wird SW1 auf 1 geschaltet, dann hält der Durchlauf an, und das gerade aktuelle Würfelbild wird ausgegeben. Wird SW1 wieder auf 0 gelegt, läuft der Würfel weiter.
Wird SW2 auf 1 gesetzt, so werden alle von da an erwürfelten Resultate addiert und auf der 7-Segment-Anzeige ausgegeben.

Zur Hardware:

Im Schaltbild ist schon angegeben, wie die 7 Leuchtdioden mit den Anschlüssen von Port A des 8155 Nr. 2 zu verbinden sind. Diese Zuordnung ist hier noch einmal übersichtlich wiedergegeben:
Wertigkeit: 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
Diode: – D6 D5 D4 D3 D2 D1 D0
Die *Initialisierung* geschieht wie bekannt mit Kommando 4:
F0 = 01, d. h. P2A ist Ausgabe- und P2C ist Eingabe-Kanal (hier sind die beiden Schalter SW1 und SW2 angeschlossen).
Nach dem Initialisierungs-Befehl müssen alle Dioden leuchten, da sie mit 0-Wert hellgesteuert werden (wenn nicht alle leuchten, muß der Fehler in der Schaltung gesucht und beseitigt werden).
Die Leuchtdioden können mit Hilfe von Kommando 4 auch einzeln angesteuert werden, und zwar mit den nachfolgend aufgeführten Werten, die jeweils

auf Adresse F1 auszugeben sind. Zur Verdeutlichung ist jeder Wert auch in binärer Schreibweise angegeben.

FE (1111 1110): D0 leuchtet
FD (1111 1101): D1 leuchtet
FB (1111 1011): D2 leuchtet
F7 (1111 0111): D3 leuchtet
EF (1110 1111): D4 leuchtet
DF (1101 1111): D5 leuchtet
BF (1011 1111): D6 leuchtet

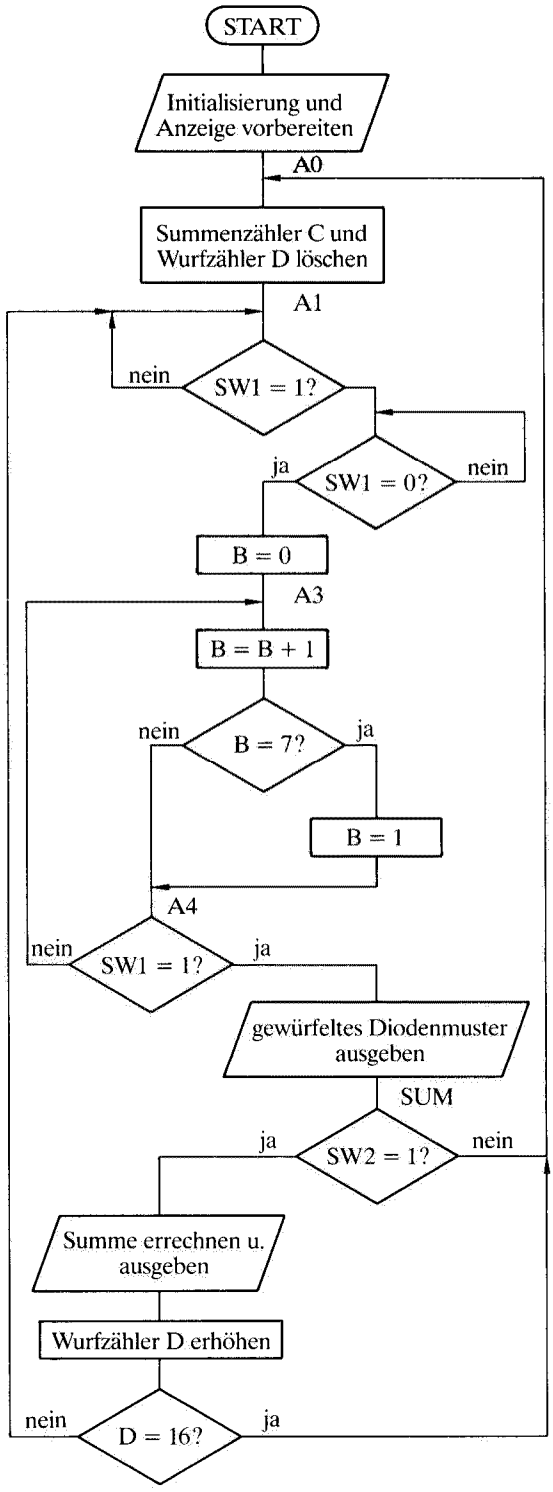
Überprüfung der Schalter:

Die beiden Schalter SW1 und SW2 sind, wie das Schaltbild zeigt, auf die beiden niederwertigsten Bits von P2C gelegt. Wenn alle anderen Bits jeweils den Wert 0 führen, dann sind mit Kommando 5 auf Adresse F3 folgende Werte zu erhalten:

F3	SW2	SW1
00	0	0
01	0	1
02	1	0
03	1	1

Lösung

Im Anschluß an das schon kurz besprochene Flußdiagramm ist das Programm aufgelistet, mit dem die oben aufgestellten Forderungen erfüllt werden.



Programmablaufplan zu Übung 12

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
1820	3E 01	START:	MVI A,01	
22	D3 F0		OUT F0	
24	3E 0F		MVI A,0F	
26	D3 F8		OUT F8	
28	3E FF		MVI A,FF	Initialisierung, Löschen der Anzeigen
2A	D3 FA		OUT FA	
2C	3E 1F		MVI A,1F	
2E	D3 FB		OUT FB	
30	31 00 1C		LXI SP,1C00	
33	0E 00	A0:	MVI C,00	C und D löschen
35	16 00		MVI D,00	
37	DB F3	A1:	IN F3	
39	E6 01		ANI 01	
3B	FE 01		CPI 01	
3D	C2 37 18		JNZ A1	
40	CD B0 18		CALL ZEIT	Warten bis S1 = 1
43	DB F3		IN F3	
45	E6 01		ANI 01	
47	FE 01		CPI 01	
49	C2 37 18		JNZ A1	
4C	DB F3	A2:	IN F3	
4E	E6 01		ANI 01	
52	FE 01		CPI 01	
	CA 4C 18		JZ A2	
	CD B0 18		CALL ZEIT	Warten, bis S1 = 0
	DB F3		IN F3	
	E6 01		ANI 01	
	FE 01		CPI 01	
	CA 4C 18		JZ A2	
	06 00		MVI B,00	
1863	04	A3:	INR B	
	78		MOV A,B	
	FE 07		CPI 07	
	DA 6C 18		JC A4	Register B so lange von 1 – 7 hoch-
	06 01		MVI B,01	zählen, bis S1 wieder auf 1 ist
186C	DB F3	A4:	IN F3	
	E6 01		ANI 01	
	FE 01		CPI 01	
	C2 63 18		JNZ A3	
	CD B0 18		CALL ZEIT	
	DB F3		IN F3	
	E6 01		ANI 01	Entprellen von S1
	FE 01		CPI 01	
	C2 63 18		JNZ A3	
1881	21 C1 18	HALT:	LXI H,ADR	
	7D		MOV A,L	
	80		ADD B	richtiges Bitmuster für die
	6F		MOV L,A	Dioden ausgeben
	7E		MOV A,M	
	D3 F1		OUT F1	

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
188A	DB F3	SUM:	IN F3	
	E6 02		ANI 02	S2 = 1?
	FE 02		CPI 02	
	C2 33 18		JNZ A0	nein → neu würfeln
	14		INR D	
	79		MOV A,C	
	80		ADD B	Augenzahl in Reg. C addieren
	27		DAA	und ausgeben
	4F		MOV C,A	
	CD C8 18		CALL AUS	
	7A		MOV A,D	
	FE 10		CPI 10	Wenn schon 16mal gewürfelt, mit
	DA 37 18		JC A1	Summenbildung, dann neu
18A1	C3 33 18		JMP A0	
18B0	E5	ZEIT:	PUSH H	
	D5		PUSH D	
	16 FA		MVI D,FA	
	2E DB	E1:	MVI L,DB	
	2D	E2:	DCR L	
	C2 B6 18		JNZ E2	Entprellzeit 1/4 Sek.
	15		DCR D	
	C2 B4 18		JNZ E1	
	D1		POP D	
	E1		POP H	
C0	C9		RET	
18C1	00	ADR:	NOP	
	FE		Muster für 1	
	BD		Muster für 2	
	BC		Muster für 3	
	A5		Muster für 4	
	A4		Muster für 5	Bitmuster für die Leuchtdioden
	81		Muster für 6	
	00	AUS:	NOP	
	D3 F9		OUT F9	
	C9		RET	

Register B enthält den gewürfelten Wert (1 . . . 6).

Register C enthält die Würfelsumme.

Register D enthält die Würfelanzahl.

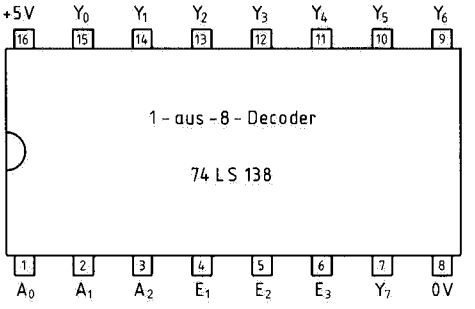
Register HL enthält die Adresse des zu B gehörenden Musters.

Übung 13: Der ECB 85 als Baustein-Prüfgerät

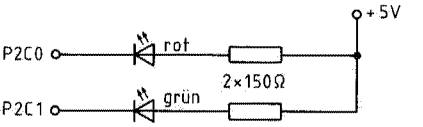
Als Beispiel dafür, daß man den ECB 85 mit seinen Peripherie-Anschlüssen auch zum Prüfen von Logik-Gattern und anderen ICs einsetzen kann, soll hier die Prüfung eines *1-aus-8-Decoders* programmiert werden. Dieser Decoder kann ausführlicher auch als *(3-Bit-Dual)-zu-(1-aus-8)-Codierer* genannt werden; aus dieser Benennung geht deutlich hervor, daß er 3 Daten-Eingänge hat (A0, A1, A2) und 8 Daten-Ausgänge (Y0, Y1, ..., Y7). Seine sogen. Wahrheitstabelle und sein Anschlußplan (von oben gesehen) sind in den Abbildungen dargestellt. In der Wahrheitstabelle ist auch angegeben, wie die genannten Ein- und Ausgänge des Decoders mit den Port-Anschlüssen des ECB 85 verschaltet werden. Die sogen. Enable-Eingänge für die normale Funktion sind wie folgt zu belegen: E1 und E2 auf L-Pegel (0 V), E3 auf H-Pegel (+5 V).

Aufgabe:

Es ist ein Programm für den ECB 85 zu schreiben, das mit Hilfe der in der Abbildung gezeigten Anwender-Schaltung den Test des genannten Decoders gestattet. Bei erfolgreichem Test soll anschließend die grüne Diode leuchten, bei einem Fehler dagegen die rote. Der Programmstart soll durch Betätigen einer beliebigen Taste des ECB-Tastenfeldes geschehen, und ebenso soll ein weiterer Programmlauf zu beginnen sein. Es soll zunächst ein möglichst detailliertes Flußdiagramm entworfen werden.
Wenn das Programm fehlerfrei gelaufen ist (mit einem intakten Decoder-IC), dann ist ein fehlerhafter Decoder zu simulieren (z. B. die Verbindung zwischen P2B2 und Anschluß A2, Pin 3, des ICs unterbrechen). Weitere Hinweise hierzu werden später gegeben.



Der Anschlußplan des Decoder-Bausteins 74LS138; vgl. Wahrheitstabelle

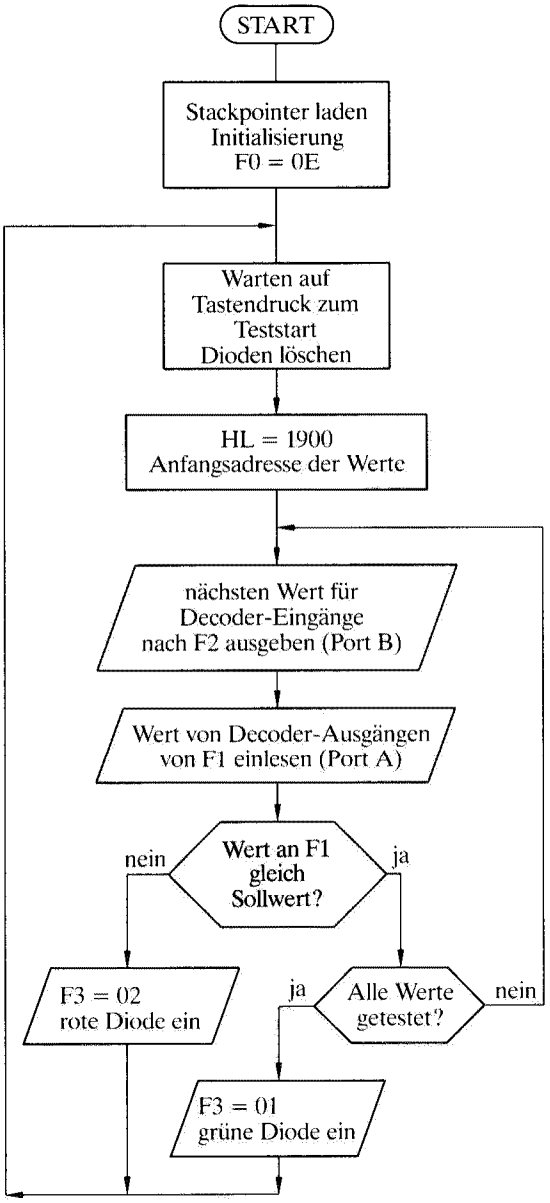


Die Anwender-Schaltung für die beiden Leuchtdioden am Port C des 8155 Nr. 2

Initialisierung: F0 = 0E;
Funktionstest der Dioden:
beide Dioden an: F3 = 00
grüne Diode an: F3 = 01
rote Diode an: F3 = 02
beide Dioden aus: F3 = 03.

Eingänge						Ausgänge								Hex.-Code
E_1	E_2	E_3	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	
1	X	X	X	X	X	1	1	1	1	1	1	1	1	FF
X	1	X												
X	X	0												
0	0	1	0	0	0	1	1	1	1	1	1	1	0	FE
			0	0	1	1	1	1	1	1	1	0	1	FD
			0	1	0	1	1	1	1	1	0	1	1	FB
			0	1	1	1	1	1	1	0	1	1	1	F7
			1	0	0	1	1	1	0	1	1	1	1	EF
			1	0	1	1	1	0	1	1	1	1	1	DF
			1	1	0	1	0	1	1	1	1	1	1	BF
			1	1	1	0	1	1	1	1	1	1	1	7F
0V	0V	+5V	B2	B1	B0	A7	A6	A5	A4	A3	A2	A1	A0	

Wahrheitstabelle des Decoder-Bausteins 74LS138. In der untersten Zeile sind die Port-Bits des 8155 Nr. 2 angegeben, mit denen die Ein- bzw. Ausgänge des Decoders zu verbinden sind. Am ECB 85 sind also auf Port P2B auszugeben: 000, 001, ..., 111, und es wird auf P2A dann eingelesen: 1111 1110, 1111 1101, ..., 0111 1111.
In der rechten Spalte ist die jeweilige am Port A zu erwartende Bit-Kombination im Hex.-Code angegeben.



Programmablaufplan (Flußdiagramm) zur Bausteinprüfung

Weitere Hinweise zur Lösung:

Es wurde weiter oben gefordert, auch ein fehlerhaftes IC zu simulieren. Wird also z. B. Anschluß 3 offen gelassen (dies wirkt bei TTL-ICs wie auf H-Pegel, logische 1, gelegt), dann tritt ein Fehler schon bei der ersten zu prüfenden Bit-Kombination 000 auf, die wegen des offenen Eingangs nun 100 lautet. Die von den Decoder-Ausgängen abgegebene Bit-Kombination lautet also 1110 1111 und nicht 1111 1110.

Simuliert man einen Fehler dadurch, daß einer der Ausgänge nicht mit dem ECB-Port verbunden wird, so kann der Fehler erst bei derjenigen Eingangs-Bit-Kombination auftreten, bei der der betr. Ausgang vom Decoder auf logisch 0 (L-Pegel) gelegt wird, die dann vom ECB nicht eingelesen werden kann.

Ähnlich wie hier gezeigt, können natürlich auch andere Halbleiter-ICs mit Hilfe des ECB 85 und seiner 8155-Ports getestet werden. Sind von einem Typ nur wenige ICs zu testen, dann kann dies auch „von Hand" mit den Kommandos 4 und 5 durchgeführt werden.

Ab Adresse 1900 werden die Eingangswerte und die zu erwartenden Ausgangswerte abgelegt:

Adresse	Hex.-Code	
1900	00	1. Eingangs-Wert
1	FE	1. Ausgangs-Wert
2	01	2. Eingangs-Wert
3	FD	2. Ausgangs-Wert
4	02	3. Eingangs-Wert
5	FB	3. Ausgangs-Wert
6	03	4. Eingangs-Wert
7	F7	4. Ausgangs-Wert
8	04	5. Eingangs-Wert
9	EF	5. Ausgangs-Wert
A	05	6. Eingangs-Wert
B	DF	6. Ausgangs-Wert
C	06	7. Eingangs-Wert
D	BF	7. Ausgangs-Wert
E	07	8. Eingangs-Wert
F	7F	8. Ausgangs-Wert

Bausteinprüfprogramm

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
1850	31		LXI SP,1C00	
1	00			
2	1C			

Adresse	Hex.-Code	Labels	Mnemo-Code	Bemerkungen
3	3E		MVI A,0E	Initialisierung
4	0E			
5	D3		OUT F0	
6	F0			
7	CD	NEU:	CALL ANWEIN	auf Tastendruck warten
8	D1			
9	07			
A	3E		MVI A,FF	Dioden löschen
B	FF			
C	D3		OUT F3	
D	F3			
E	21		LXI H,1900	Anfangsadresse
F	00			der Werte
60	19			laden
1	7E	WERT:	MOV A,M	Eingangskombination
2	D3		OUT F2	ausgeben
3	F2			
4	23		INX H	
5	DB		IN F1	Ausgangs-Information
6	F1			mit Soll-Information
7	BE		CMP M	vergleichen
8	CA		JZ WEI	
9	72			
A	18			
B	3E		MVI A,02	Fehler:
C	02			rote Diode an
D	D3		OUT F3	
E	F3			
F	C3		JMP NEU	
70	57			
1	18			
2	7D	WEI:	MOV A,L	Alle 8 Werte
3	FE		CPI 0F	untersucht?
4	0F			
5	CA		JZ FERTIG	
6	7C			
7	18			
8	23		INX H	
9	C3		JMP WERT	
A	61			
B	18			
C	3E	FERTIG:	MVI A,01	grüne Diode an
D	01			
E	D3		OUT F3	
F	F3			
80	C3		JMP NEU	
1	57			
2	18			

Teil 3: Anhang

A.1 Aufbau eines kleinen Mikrocomputersystems

1. Prozessorplatine

Die Prozessorplatine wird mit folgenden Bausteinen aufgebaut:

1 SAB 8085 (Mikroprozessor)
2 SAB 8155 (Ein-Ausgabebausteine mit Timer und RAM)
2 SAB 2716 (EPROMs, 2K bzw. 4K)
oder
2 SAB 2732
alternativ zu den beiden EPROMs auch 2K EPROM (2716) und 2K RAM (6116); die hierfür nötigen Änderungen sind beim Stromlaufplan angegeben.

Als Zusatzlogik werden benötigt:

1 SN 74 LS 373 (zum Separieren der Adressenbits $A_0 - A_7$)
1 SN 74 LS 138 (zur Erzeugung der CS-Signale für die Speicher- und E/A-Bausteine)

Ferner benötigt man

1 Widerstand 560 kΩ
1 Diode
1 Kondensator der Kapazität 100 nF
5 Widerstände zu 10 kΩ
1 Widerstand zu 4,7 kΩ
1 Quarz 6,144 MHz

Es werden alle wichtigen Prozessorsignale auf Stiftleisten geführt, so daß eine Erweiterung des Systems jederzeit möglich ist. Auch die Ein-Ausgabe-Ports und die Timer-Ein-/Ausgänge sind auf Stiftleisten geführt.
Die Verschaltung der einzelnen Bausteine ist dem Stromlaufplan zu entnehmen; die Anordnung der Bausteine ist aus der Layout-Kopie ersichtlich.

Testen der Prozessor-Platine

Es gibt grundsätzlich zwei Arten, ein solches System zu testen.

1. Mit Hilfe eines Adapters und des ECB 85 (Prozessor-Platine noch ohne 8085).
2. Nur im System selbst durch Stecken des Prozessors 8085 und eines EPROMs mit entsprechendem Prüfprogramm.

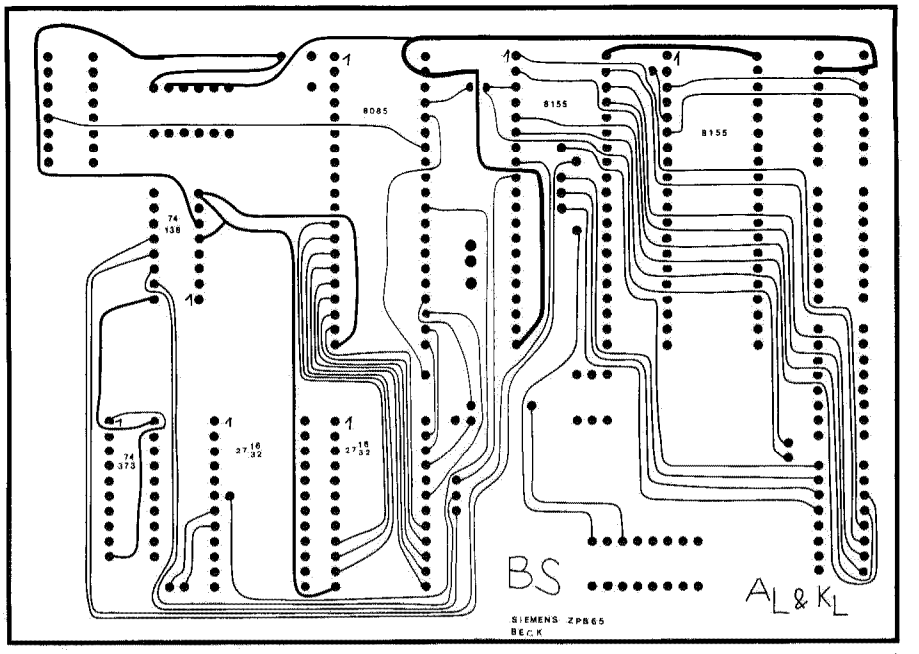
Beide Arten sollten zum Testen verwendet werden, um sicher zu gehen, daß der einzusetzende Prozessor auch in Ordnung und somit das System voll funktionsfähig ist.

Das Testen erfolgt mit Hilfe eines Testprogrammes, das folgende Funktionen überprüfen sollte:

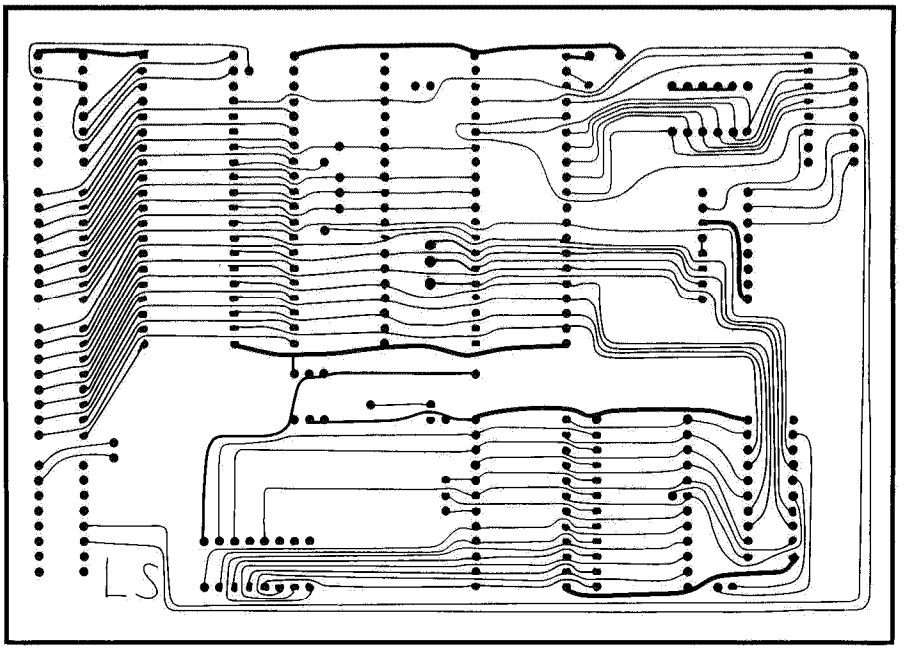
— die Timer-Ausgänge der E/A-Bausteine 8155
— die Port-Ausgänge der E/A-Bausteine 8155

Beispiel für ein Testprogramm

— Baustein 8155 Nr. 0 initialisieren
— Am Timer-Ausgang vom 8155 Nr. 0 einen Rechteckpuls ausgeben (man kann dort einen Lautsprecher anschließen, um diesen Ton hörbar zu machen)
— alle drei Ports des 8155 als Ausgabeports deklarieren
— an allen Portbits nacheinander eine 0 sowie eine 1 ausgeben (diese Signale können an den zu den Verbindungssockeln hingeführten Leitungen mit Hilfe eines Logiktesters abgetastet werden)
— Baustein 8155 Nr. 1 initialisieren
— weiterer Testvorgang analog zu Baustein Nr. 0



Layout der Prozessorplatine (Bestückungsseite).



Layout der Prozessorplatine (Leiterbahnseite).

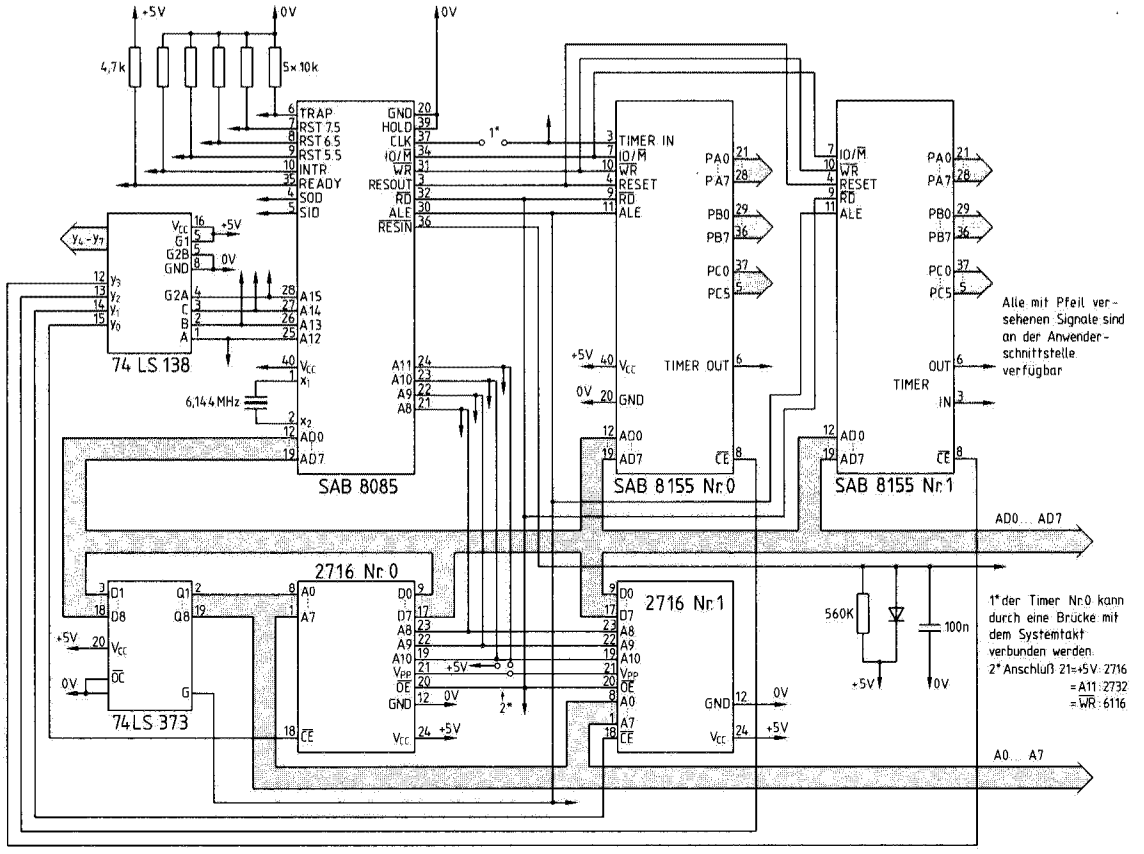
Zu 1.:

Das Testprogramm muß auf einem EPROM vom Typ 2716 oder 2732 ab Adresse 1000H ablauffähig stehen. Dieses EPROM wird dann auf das System gesteckt (siehe Stromlaufplan); als nächstes muß das System mit dem ECB 85 über den Adapter verbunden werden. Jetzt erst kann das Testprogramm mit Kommando 7, 8 oder 9 des ECB 85 gestartet werden.
Der Adapter ist ein 40poliger Stecker, der die Prozessorsignale des 8085 auf dem ECB auf das Testsystem überträgt und den Prozessor im Testsystem simuliert.

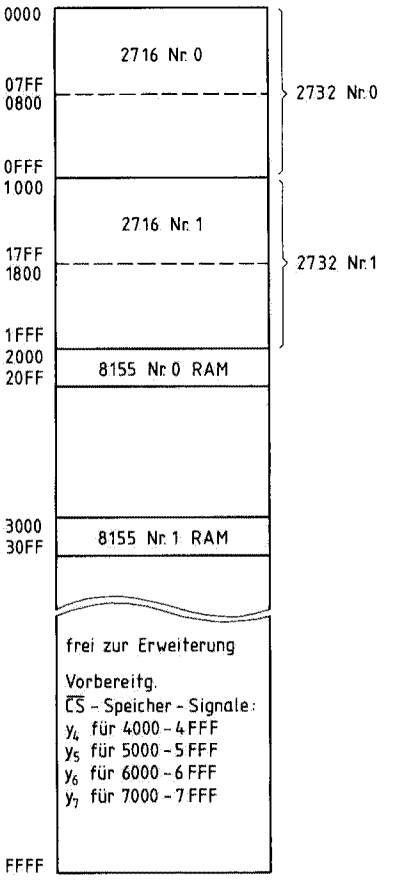
Auch die Kommandos 4 und 5 können benutzt werden, um versteckte Fehler zu lokalisieren.

Zu 2.:

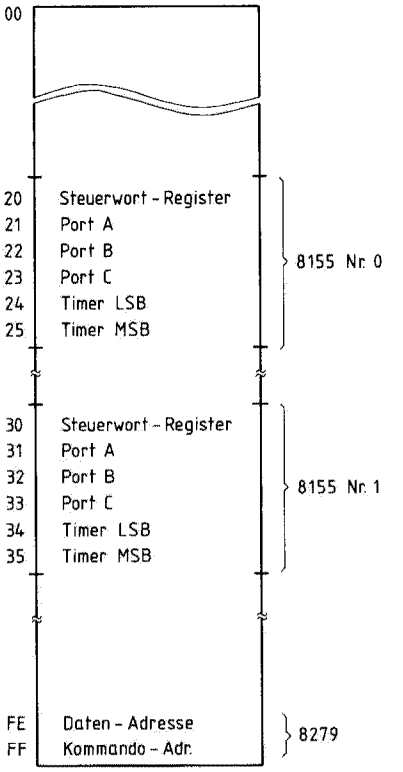
Will man die Platine des Minimalsystems ohne Adapter testen, so muß ein 8085 gesteckt werden. Außerdem muß das Testprogramm (s. u.) ab Adresse 0000H ablauffähig sein. Gesteckt wird es laut Stromlaufplan. Sobald nun die Spannung eingeschaltet wird, läuft das Programm ab.



Stromlaufplan der Prozessorplatine. Alle mit Pfeil versehenen Signale sind an der Anwender-Schnittstelle verfügbar. Der Timer des 8155 Nr. 0 kann durch eine Brücke mit dem Systemtakt verbunden werden.
Der Anschluß Nr. 21 der Speicherbausteine ist wie folgt zu verdrahten: beim 2716: auf +5 V legen; beim 2732: mit A_{11} verbinden; beim 6116: mit $\overline{WR}$ verbinden.



Speicherbelegung.
Wenn beide EPROMs gesteckt sind, müssen sie typengleich sein. Wenn ein EPROM und ein RAM eingesetzt werden, so ist der gemeinsame Anschluß 21 aufzutrennen; vgl. hierzu die Erläuterungen beim Stromlaufplan. $Y_4 - Y_7$ sind von außen abgreifbare Ausgänge des (1-aus-8)-Decoders 74LS138.



Ein-/Ausgabe-Adreßbereich von Prozessor- und Anzeige-Platine.

Prüfung und Inbetriebnahme der Minimalsystemplatine

Zur Prüfung der Minimalsystemplatine benützt man am günstigsten ein kleines Programm, das in dem EPROM 2716 Nr. 0 ab Adresse 0000H steht. Da beim Rücksetzen des Prozessors der Programmzähler mit der Adresse 0000H geladen wird, starten wir das Programm, indem wir die Spannung einschalten. Das Programm initialisiert die beiden auf der Platine befindlichen E/A-Bausteine 8155 und gibt an alle Portbits einen Reckteckpuls aus, der durch einen Lautsprecher hörbar gemacht wird.

Testprogramm

Adresse	Hex.-Code	Mnemo-Code	Bemerkungen
0000	C3	JMP 0050	
1	50		
2	00		
0050	31	LXI SP,2100	
1	00		
2	21		
3	3E	MVI A,0F	alle Ports
4	0F		sind
5	D3	OUT 20	Ausgabeport
6	20		
7	D3	OUT 30	Ausgabe des
8	30		Akkuinhalts
9	3E	MVI A,00	auf
A	00		allen Ports
B	D3	OUT 21	
C	21		
D	D3	OUT 22	
E	22		
F	D3	OUT 23	
60	23		
1	D3	OUT 31	
2	31		
3	D3	OUT 32	
4	32		
5	D3	OUT 33	
6	33		
7	CD	CALL 0100	
8	00		
9	01		
A	3C	INR A	
B	C3	JMP 005B	
C	5B		
D	00		
0100	06	MVI B,1E	
1	1E		
2	05	DCR B	diese Zeit-
3	C2	JNZ 0102	schleife rea-
4	02		lisiert eine
5	01		Zeit von
6	C9	RET	0,125 ms.

Dieses Programm erzeugt an allen Portausgängen einen Rechteckpuls mit folgenden Frequenzen (die über einen Lautsprecher hörbar zu machen sind, der über einen 10-Ω-Widerstand an den betr. Ausgang geschaltet wird):

An A0, B0 und C0: 4 kHz
an A1, B1 und C1: 2 kHz
an A2, B2 und C2: 1 kHz
an A3, B3 und C3: 500 Hz
an A4, B4 und C4: 250 Hz
an A5, B5 und C5: 125 Hz
an A6 und B6: 62,5 Hz
an A7 und B7: 31,25 Hz

Ein erfolgreicher Test läßt die Aussage zu, daß das System funktionsfähig ist.

2. Anzeige- und Tastatur-Platine

Die Anzeige- und Tastatur-Platine wird mit folgenden Bausteinen bestückt:

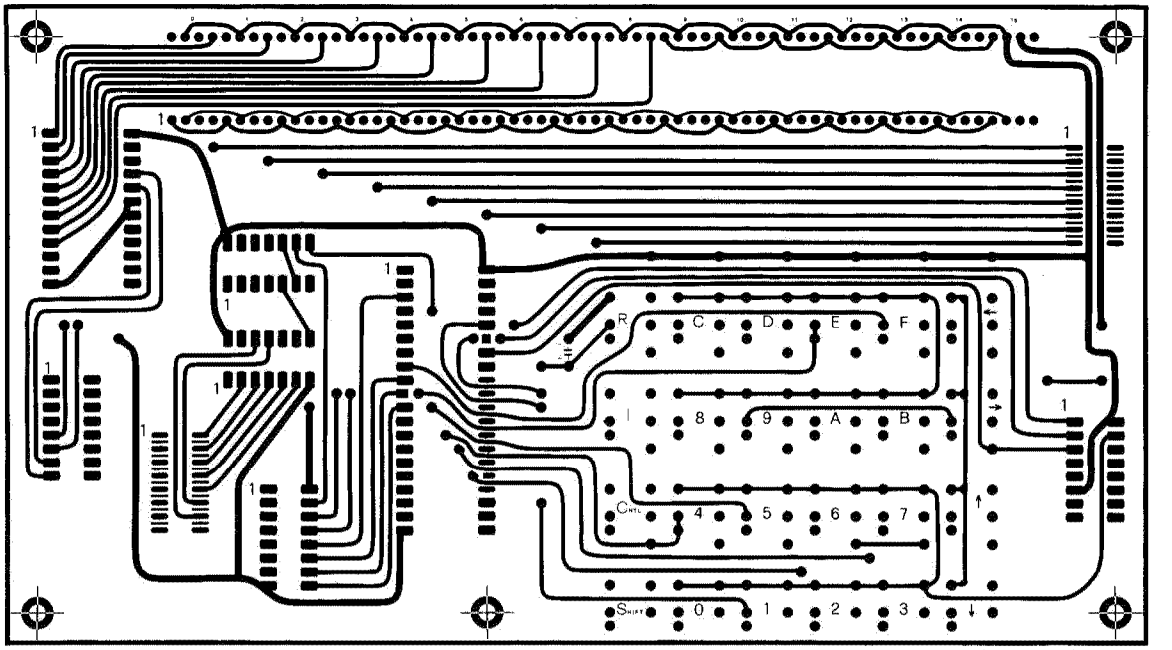
1 SN 74154 (1-aus-16-Decoder)
1 SAB 8279 (Tastatur- und Anzeige-Steuerbaustein)
1 SN 7430 } (Zur Erzeugung des $\overline{CS}$-
1 SN 74LS00 } Signals)
1 SN 74LS138 (1-aus-8-Decoder zur Steuerung der Tastatur)

16 Anzeigen
HP5082-7340
24 Tasten

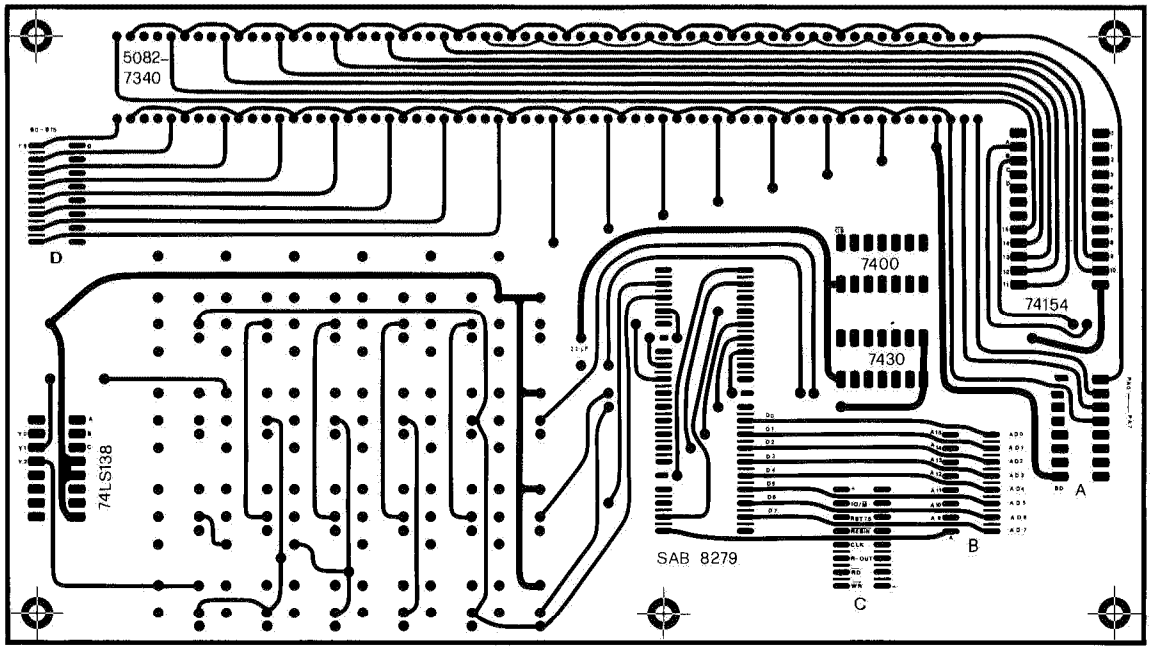
Die Verschaltung der einzelnen Bauteile entnehmen Sie bitte dem Stromlaufplan. Die Anordnung der Bauteile ist aus dem Layout zu ersehen.

Test der Tastatur-/Anzeige-Platine

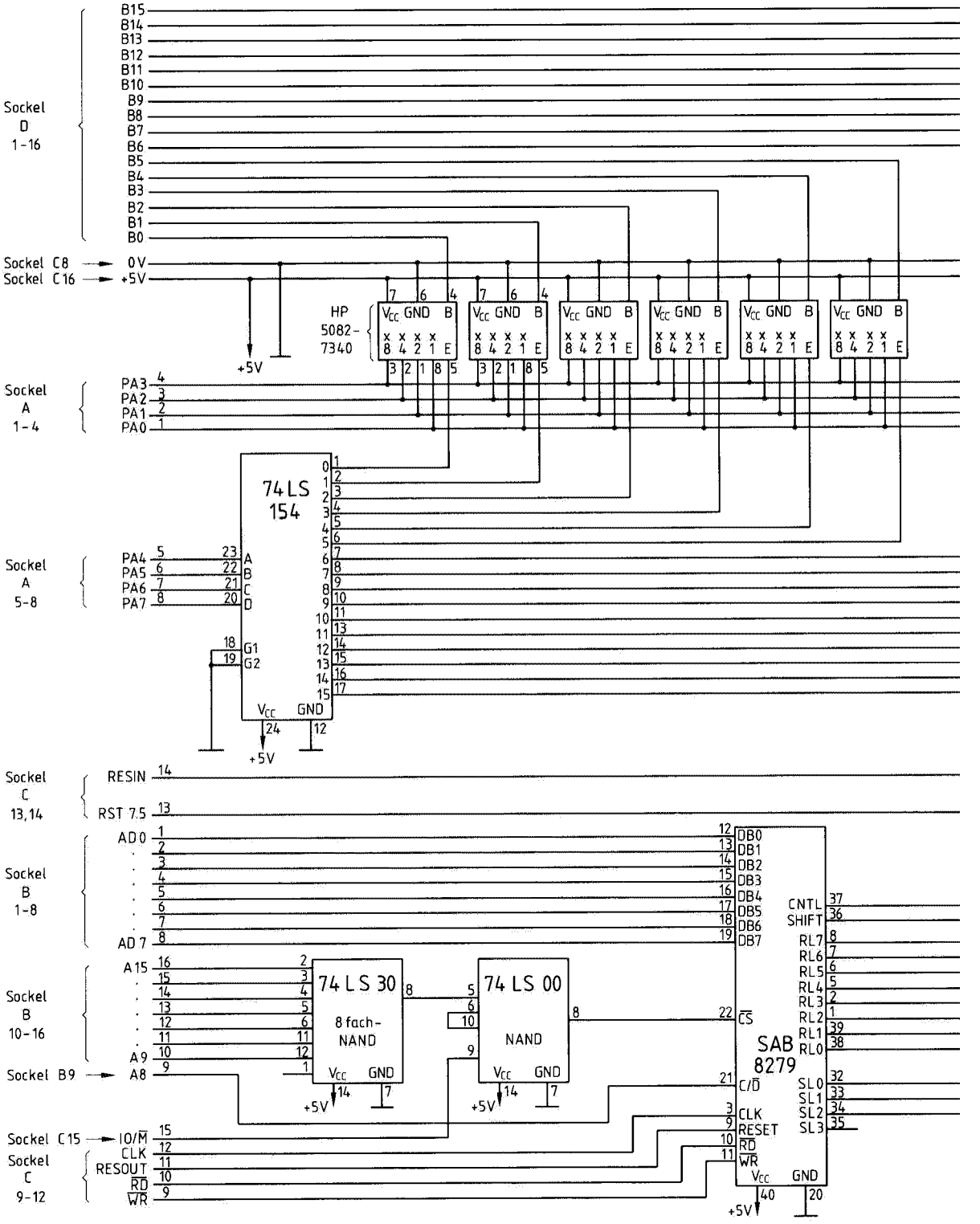
Um die Anzeige- und Tastatur-Platine auf ihre richtige Funktion prüfen zu können, schließen wir die fertig bestückte Platine über eine 96polige Messerleiste an den ECB 85 an.
Dazu werden die an den Sockeln A, B, C und D erforderten Signale, wie aus dem Stromlauf ersichtlich, angelegt. Die Daten zur Anzeige sind nun über den Port A Bit 0 bis 3 zuzuführen, die Anzeigenstellenauswahl wird über das höherwertige Halbbyte des Ports A durchgeführt, das über den 1-aus-16-Decoder jeweils eine Anzeige auswählt. Die Dunkelsteuerung erfolgt über die Ports B, wobei jedes Bit des Ports B einer Stelle zugeordnet ist.
Die Tastaturabfrage erfolgt über den SAB 8279, der zu seiner Funktion die Daten und einige Steuersignale benötigt. Für die Bausteinauswahl müssen die Adressen A_0 sowie $A_9 - A_{15}$ zugeführt werden.

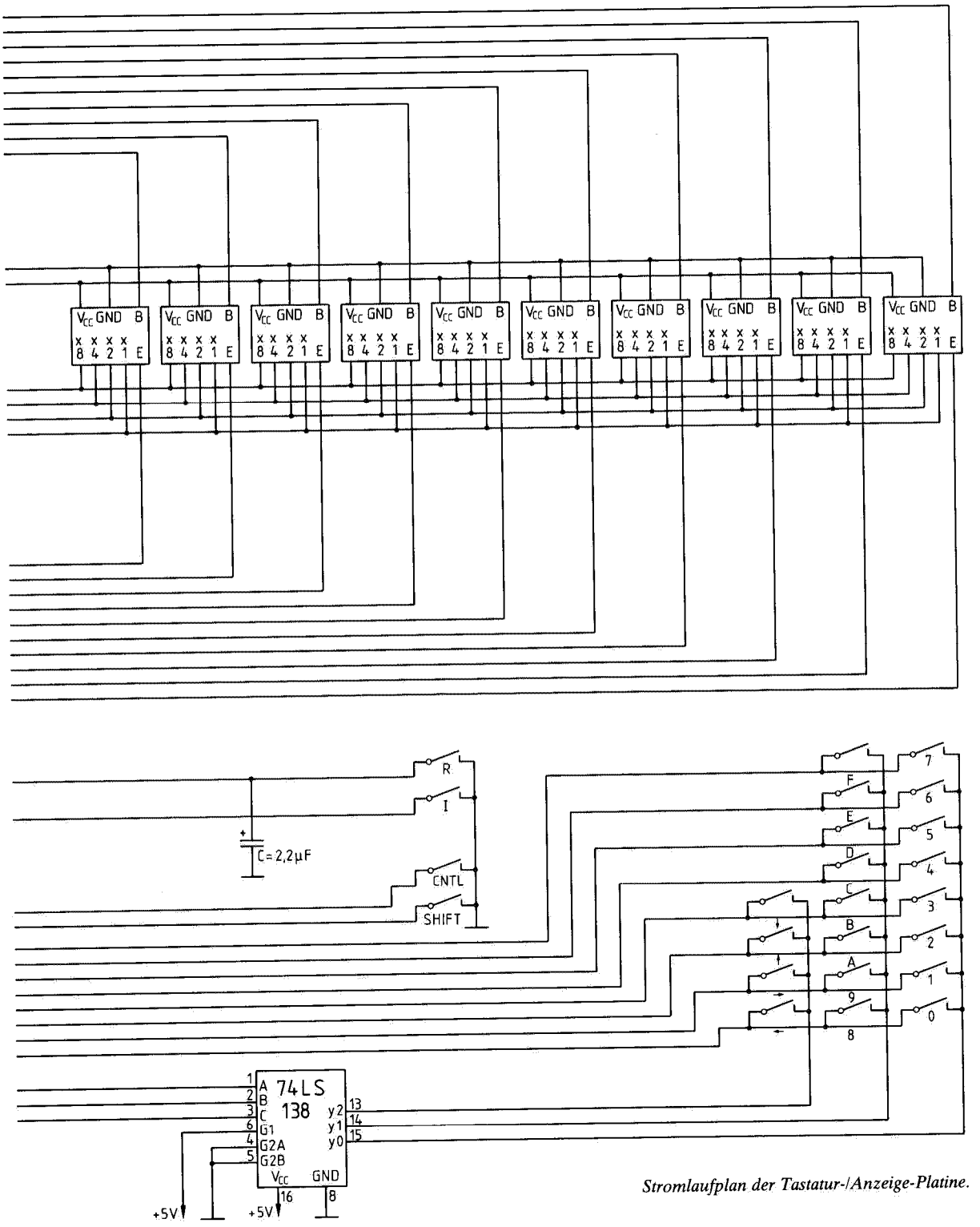


Layout der Tastatur-/Anzeige-Platine (Bestückungsseite).



Layout der Tastatur-/Anzeige-Platine (Leiterbahnseite).


Sockel D 1-16
B15
B14
B13
B12
B11
B10
B9
B8
B7
B6
B5
B4
B3
B2
B1
B0
Sockel C8
Sockel C16
0V
+5V
HP 5082-7340
Vcc GND B
x x x x x
8 4 2 1 E
Sockel A 1-4
PA3
PA2
PA1
PA0
74LS 154
Sockel A 5-8
PA4
PA5
PA6
PA7
A
B
C
D
G1
G2
Vcc
GND
Sockel C 13,14
RESIN
RST 7.5
Sockel B 1-8
AD0
AD7
Sockel B 10-16
A15
A9
Sockel B9
A8
74 LS 30
8 fach-NAND
74 LS 00
NAND
Sockel C15
IO/M
Sockel C 9-12
CLK
RESOUT
RD
WR
SAB 8279
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
CS
C/D
CLK
RESET
RD
WR
CNTL
SHIFT
RL7
RL6
RL5
RL4
RL3
RL2
RL1
RL0
SL0
SL1
SL2
SL3



Stromlaufplan der Tastatur-/Anzeige-Platine.

Bei den Adreßsignalen ist A_{15} durch $\overline{A15}$ zu ersetzen, so daß die E/A-Adresse des 8279-Bausteins für die Daten 7E ist, und für die Kommandos 7F. (Im eigenen System kann A_{15} verdrahtet werden, so daß dann die Adressen FE und FF lauten. In Zusammenarbeit mit dem ECB 85 aber würde das zu Schwierigkeiten führen, da der Baustein 8155 Nr. 1 des ECB 85 mit den Adressen F8 bis FF angesprochen wird.)
Da wir mit dem 8279 nur die Tastatur steuern, brauchen wir beim Betriebsartenwort nur auf die richtige Tastenfunktion zu achten.
Die Tastatur soll im sogen. codierten 2-key-lockout-Betrieb (des 8279) arbeiten; also müssen wir am ECB eingeben:

4. PA = 7F ↓
4. 7F = 00 ↓

Jetzt wollen wir die interne Verarbeitungsgeschwindigkeit festlegen:

4. 7F = 3F ↓ Interne Arbeitsfrequenz 100 kHz

Nun müssen wir überprüfen, ob der 8279 auch unsere angeschlossenen 24 Tasten richtig auswertet. Dazu müssen wir ihm sagen, daß wir in den sogen. FIFO schreiben wollen. Zuvor aber löschen wir den FIFO mit dem Kommando

4. 7F = C2 ↓
4. 7F = 40 ↓ Löschen des FIFOs sagt dem Baustein, daß wir den FIFO lesen wollen

Nun können wir bis zu 8 Tasten nacheinander betätigen und mit Kommando 5 den FIFO lesen. Nehmen wir an, wir haben nur die Tasten 0, 1, 2, 3, 4, 5, 6, 7 betätigt und fragen nun ab:

5. PA = 7E ↓
5. 7E = C0 ↓,

dann erhalten wir mit jedem Quittungssignal:

5. 7E = C1 ↓
5. 7E = C2 ↓
5. 7E = C3 ↓
5. 7E = C4 ↓
5. 7E = C5 ↓
5. 7E = C6 ↓
5. 7E = C7 ↓

Das ist z. B. bei C1 folgendes Bitmuster

$$\underset{\text{Cntr}}{1}\ \underset{\text{Shift}}{1}\ \underbrace{0\,0\ \ 0\,0\,0\,1}_{\text{Tastenposition}}$$

Drücken wir jetzt gleichzeitig Cntr und 1, dann erhalten wir:

5. 7E = 41 ↓;

drücken wir Shift und 1, dann bekommen wir:

5. 7E = 81 ↓.

Auf gleiche Weise können nun noch die anderen Tasten geprüft werden.
Ist alles wie erwartet gelaufen, dann kann man sagen, der 8279 arbeitet in der gewünschten Betriebsart einwandfrei.
Nun ist noch die 16stellige Anzeige auf ihre richtige Funktion zu prüfen.
Da die Information über Anzeigeninhalt und Anzeigestelle über den Port A kommt, sowie die Dunkelsteuerung über die Ports B der beiden E/A-Bausteine, müssen die 8155-Bausteine wie folgt initialisiert werden:

4. F8 = 03 ↓ Port A und Port B sind Ausgabeports
4. F0 = 02 ↓ Port B ist Ausgabeport

Wenn alle Anzeigen beschreibbar sein sollen, dann müssen die Ausgangssignale der Ports B Null sein. Dies wird erreicht mit:

4. FA = 00 ↓
4. F2 = 00 ↓

Nun soll getestet werden, ob die Dunkelsteuerung funktioniert; dazu geben wir ein:

4. F2 = FF ↓

(Wir steuern damit die Anzeigen 8–16 dunkel.)

4. FA = 7F gibt nur die 1. Stelle frei
4. FA = BF gibt nur die 2. Stelle frei
4. FA = DF gibt nur die 3. Stelle frei
4. FA = EF gibt nur die 4. Stelle frei
4. FA = F7 gibt nur die 5. Stelle frei
4. FA = FB gibt nur die 6. Stelle frei
4. FA = FD gibt nur die 7. Stelle frei
4. FA = FE gibt nur die 8. Stelle frei

Der Test der Anzeigen 8–16 erfolgt in gleicher Weise; hierbei ist dann FA = FF.

Danach ist zu überprüfen, ob an den einzelnen Stellen die gewünschte Information ausgegeben werden kann.
Dazu geben wir die betreffende Stelle über Port B frei und schreiben die gewünschte Information ein; z. B.:

4.	F2	=	FF	↓	
4.	FA	=	EF	↓	(4. Stelle kann beschrieben werden)
4.	F9	=	33	↓	(schreibt in die 4. Stelle eine 3)

Analog können alle Anzeigen getestet werden.
Wenn auch die Anzeigeplatine funktioniert, kann man sich daran machen, beide Platinen zu einem vollständigen System zusammenzuschalten. Die genaue Verschaltung entnehmen Sie bitte den beiden Stromlaufplänen.
Die Kabel für die Verbindung von Prozessorplatine und Tastaturplatine werden aus 16poligem Kabel und Endlosverbindern gefertigt. Es ist sinnvoll, diese vor ihrem Einsatz zu testen. Das kann z. B. auf einfache Weise mit der Flagbyteanzeige durchgeführt werden, auf der 8 voneinander unabhängige Dioden hell oder dunkel gesteuert werden können (vgl. *Übung 4*). Testen Sie, ob jede einzelne Diode gesteuert werden kann. Ist das der Fall, dann ist Ihr Kabel in Ordnung.
Das System können Sie dann nach erfolgtem Zusammenbau mit dem im folgenden beschriebenen Testprogramm testen.

Beschreibung des nachfolgend abgedruckten Testprogramms:

Nach dem Rücksetzen bzw. nach dem Einschalten des Systems blinkt die 1. Stelle links.
Nun kann durch Drücken einer der Tasten 0–F bzw. der Funktionstaste ↓ folgendes erreicht werden:
Wird eine der Tasten 0–F gedrückt, dann wird das der Taste entsprechende Zeichen angezeigt und die Blinkstelle um 1 Stelle nach rechts verschoben. Bei Druck der Taste ↓ wartet das System auf einen Druck der Tasten 0–F und bringt dann das der Taste entsprechende Zeichen auf allen Anzeigen so oft blinkend zur Anzeige, wie das Zeichen angibt.
Z. B. ein Druck von ↓ und anschließend von 9 bewirkt, daß die 9 neunmal auf allen 16 Anzeigen blinkt.
Danach befindet sich das System in rückgesetztem Zustand.

Testprogramm:

Adresse	Hex.-Code	Labels	Mnemo-Code
00 00	31 00 31	START:	LXI SP,3100
03	3E 0F		MVI A,0F
05	D3 30		OUT 30
07	D3 20		OUT 20
09	3E 00		MVI A,00
0B	D3 21	X1:	OUT 21
0D	C6 10		ADI 10
0F	FE 00		CPI 00
11	C2 0B 00		JNZ X1
14	D3 21		OUT 21
16	32 F0 30		STA 30F0
19	3E FF		MVI A,FF
1B	32 F1 30		STA 30F1
1E	32 F2 30		STA 30F2
21	D3 22		OUT 22
23	D3 32		OUT 32
25	3E 00		MVI A,00
27	D3 FF		OUT FF
29	3E C2		MVI A,C2
2B	D3 FF		OUT FF
2D	01 00 40	BEDIEN:	LXI B,4000
30	DB FF	TAST:	IN FF
32	E6 07		ANI 07
34	C2 42 00		JNZ CLOE
37	0D		DCR C
00 38	C2 30 00		JNZ TAST
3B	05		DCR B
3C	C2 30 00		JNZ TAST
3F	C3 00 01		JMP BLINK
42	DB FE	CLOE:	IN FE
44	E6 3F		ANI 3F
46	4F		MOV C,A
47	C3 87 00		JMP AUSW
00 50	C3 50 01	FEHLAD:	JMP BZAHL
00 87	FE 10	AUSW:	CPI 10
89	CC FF FF		CZ XX XX
8C	FE 11		CPI 11
8E	CC FF FF		CZ XX XX
91	FE 12		CPI 12
93	CC FF FF		CZ XX XX
96	FE 13		CPI 13
98	CC 50 00		CZ FEHLAD
9B	00		NOP
9C	3A F0 30		LDA 30F0
9F	81		ADD C
A0	D3 21		OUT 21
A2	C6 10		ADI 10
A4	D3 21		OUT 21
A6	E6 F0		ANI F0
A8	D3 21		OUT 21
AA	32 F0 30		STA 30F0
AD	3A F1 30		LDA 30F1
B0	FE 00		CPI 00
B2	CA C0 00		JZ ZELL2

Adresse	Hex.-Code	Labels	Mnemo-Code
B5	0F		RRC
B6	E6 7F		ANI 7F
B8	32 F1 30	ZELL1:	STA 30F1
BB	D3 22		OUT 22
BD	C3 2D 00		JMP BEDIEN
C0	3A F2 30	ZELL2:	LDA 30F2
C3	0F		RRC
C4	E6 7F		ANI 7F
C6	32 F2 30		STA 30F2
C9	D3 32		OUT 32
CB	00 00		NOP
CD	00		NOP
CE	FE 00		CPI 00
D0	C2 DB 00		JNZ ZURÜ
D3	3E FF		MVI A,FF
D5	32 F1 30		STA 30F1
D8	32 F2 30		STA 30F2
DB	C3 2D 00	ZURÜ:	JMP BEDIEN
01 00	3A F0 30	BLINK:	LDA 30F0
03	E6 F0		ANI F0
05	07		RLC
06	07		RLC
07	07		RLC
08	07		RLC
09	47		MOV B,A
0A	FE 08		CPI 08
0C	D2 14 01		JNC LSBYTE
0F	DB 22		IN 22
11	C3 19 01		JMP STELNEG
14	D6 08	LSBYTE:	SUI 08
16	47		MOV B,A
17	DB 32		IN 32
19	C5	STELNEG:	PUSH B
1A	07	LVER:	RLC
1B	05		DCR B
1C	F2 1A 01		JP LVER
1F	3F		CMC
20	C1		POP B
21	1F		RAR
22	0F	RVER:	RRC
23	05		DCR B
24	C2 22 01		JNZ RVER
27	F5		PUSH PSW
28	3A F0 30		LDA 30F0
2B	E6 F0		ANI F0
2D	FE 80		CPI 80
2F	D2 38 01		JNC LSBOUT
32	F1		POP PSW
33	D3 22		OUT 22
35	C3 3B 01		JMP ZURÜ 1

Adresse	Hex.-Code	Labels	Mnemo-Code
38	F1	LSBOUT:	POP PSW
39	D3 32		OUT 32
3B	C3 2D 00	ZURÜ1:	JMP BEDIEN
01 50	DB FF	BZAHL:	IN FF
52	E6 07		ANI 07
54	CA 50 01		JZ BZAHL
57	DB FE		IN FE
59	E6 0F		ANI 0F
5B	D3 21		OUT 21
5D	C6 10	AUSG:	ADI 10
5F	D3 21		OUT 21
61	67		MOV H,A
62	E6 F0		ANI F0
64	7C		MOV A,H
65	00		NOP
66	C2 5D 01		JNZ AUSG
69	3E 00		MVI A,00
6B	D3 22		OUT 22
6D	D3 32		OUT 32
6F	01 00 A0	ZEIT:	LXI B,A000
72	0D	Z1:	DCR C
73	C2 72 01		JNZ Z1
76	05		DCR B
77	C2 72 01		JNZ Z1
7A	DB 22		IN 22
7C	FE 00		CPI 00
7E	C2 8A 01		JNZ HELL
81	3E FF		MVI A,FF
83	D3 22		OUT 22
85	D3 32		OUT 32
87	C3 6F 01		JMP ZEIT
8A	3E 00	HELL:	MVI A,00
8C	D3 22		OUT 22
8E	D3 32		OUT 32
90	01 00 A0	ZEIT2:	LXI B,A000
93	0D	Z2:	DCR C
94	C2 93 01		JNZ Z2
97	05		DCR B
98	C2 93 01		JNZ Z2
01 9B	00		NOP
9C	00		NOP
9D	00		NOP
9E	00		NOP
9F	00		NOP
A0	25		DCR H
A1	C2 6F 01		JNZ ZEIT
A4	21 00 00		LXI H,0000
A7	E3		XTHL
A8	C9		RET

A.2 Listing der in den Übungen benutzten Monitor-Unterprogramme des ECB 85.

Monitor-Unterprogramm UMW:

```
Programm      : UMW
Funktion      : UMW wandelt das niederwertige Halbbyte des Akkus in ein 7-Segmentmuster um.
Eing. Param.  : <A>  =  Hex.-Zeichen im niederw. Halbbyte
Ausg. Param.  : <A>  =  7-Segmentmuster
Veränd. Reg.  : PSW
Aufger. Prog. : –
```

```
05C9  E5        UMW:    PUSH    H
05CA  E60F              ANI     0F          ;Niederw. Halbbyte maskieren
05CC  21D805            LXI     H,HEXO      ;Lade H&L mit Tabellenadr. („HEXO")
05CF  85                ADD     L           ;addiere Hex.-Code hinzu
05D0  6F                MOV     L,A
05D1  3E00              MVI     A,00
05D3  8C                ADC     H
05D4  67                MOV     H,A
05D5  7E                MOV     A,M         ;7-Segmentmuster in Akku
05D6  E1                POP     H
05D7  C9                RET

                        ;Tabelle für Hex.-Code-Anzeige

05D8  3F        HEXO:   DB      3FH         ;7-Segmentmuster für 0
05D9  06                DB      06H         ;7-Segmentmuster für 1
05DA  5B                DB      5BH         ;7-Segmentmuster für 2
05DB  4F                DB      4FH         ;7-Segmentmuster für 3
05DC  66                DB      66H         ;7-Segmentmuster für 4
05DD  6D                DB      6DH         ;7-Segmentmuster für 5
05DE  7D                DB      7DH         ;7-Segmentmuster für 6
05DF  07                DB      07H         ;7-Segmentmuster für 7
05ED  7F                DB      7FH         ;7-Segmentmuster für 8
05E1  6F                DB      6FH         ;7-Segmentmuster für 9
05E2  77                DB      77H         ;7-Segmentmuster für A
05E3  7C                DB      7CH         ;7-Segmentmuster für B
05E4  39                DB      39H         ;7-Segmentmuster für C
05E5  5E                DB      5EH         ;7-Segmentmuster für D
05E6  79                DB      79H         ;7-Segmentmuster für E
05E7  71                DB      71H         ;7-Segmentmuster für F
```

Monitor-Unterprogramm ANWAZL:

Programm : ANWAZL (Anzeige löschen)
Funktion : Die über den Tastatur-/Anzeigebaustein SAB8279 gesteuerte Anzeige wird gelöscht (alle Anzeigen „Nullen", d.h. Dunkel).

Eing. Param. : –
Eingaben von : 8279 (Status)
Ausg. Param. : –
Ausgaben an : 8279 (Lösch-Kommando)
Veränd. Reg. : PSW
Aufger. Prog. : Anwaus
Bemerkung : Bei Verwendung des Kommandos „Anzeige löschen" kann es vorkommen, daß nicht alle Stellen gelöscht werden (Eigenschaft des 8279). Aus diesem Grund werden alle Stellen einzeln gelöscht.

```
07BC  97      ANWAZL:  SUB   A       ;Code für Zeichen „Null" laden
07BD  0608             MVI   B,08    ;B-Reg. für Anzeigestelle setzen
07BF  05      SLAZL:   DCR   B       ;Anzeigestelle dekrementieren
07C0  CDC707           CALL  ANWAUS  ;Zeichen „Null" ausgeben
                                     ;Anzeigenstelle = 0?
07C3  C2BF07           JNZ   SLAZL   ;Nein! Nächste Stelle
07C6  C9               RET           ;Ja! Rücksprung
```

Monitor-Unterprogramm ANWAUS:

Programm : ANWAUS (Ausgabe zur Anzeige)
Funktion : Der im A-Reg. übergebene 7-Segment-Code wird über den Tastatur-/Anzeigenbaustein SAB8279 zur Anzeige ausgegeben. Die Anzeige-Position (Stelle) ist im B-Reg. zu übergeben.

Eing. Param. : 7-Segment-Code in A-Reg.
Anzeigeposition im B-Reg. (0–7, wobei die Zählung von links nach rechts erfolgt)
Eingaben von : –
Ausg. Param. : –
Ausgaben an : 8279 (Anzeige-Ram)
Veränd. Reg. : Keine

```
07C7  F5      ANWAUS:  PUSH  PSW     ;/7-Segment-Code im Kellerspeicher ablegen
07C8  3E80             MVI   A,80    ;Maske für „Schreiben Anzeige-Ram 0" laden
07CA  80               ADD   B       ;Nr. der Anzeigenposition zur Maske addieren
07CB  D3EF             OUT   EF      ;Erzeugtes Kommandowort zur 8279 ausgeben
07CD  F1               POP   PSW     ;7/-Segment-Code laden
07CE  D3EE             OUT   EE      ;7-Segment-Code zur Anzeige über 8279
                                     ausgeben
                                     ;0EEH ist die Adresse des Anzeige- und des
                                     ;FIFO-RAM in 8279
07D0  C9               RET           ;Rücksprung
```

Monitor-Unterprogramm ANWEIN

Programm	: Anwein	(Eingabe von Tastatur)
Funktion	: Der Pufferspeicher (FIFO-RAM) für die Tastatureingabe wird gelöscht und gewartet bis eine Tastatureingabe erfolgt. Wird ein Zeichen eingegeben, erfolgt der Rücksprung mit Übergabe des Zeichencodes in A-Reg.	
Eing. Param.	: –	
Eingaben von	: 8279 (Status, FIFO-RAM)	
Ausg. Param.	: Zeichen im A-Reg.	
Ausgaben an	: 8279 (FIFO-RAM)	
Veränd. Reg.	: PSW	

```
07D1  3EC2      ANWEIN:  MVI   A,C2   ;„FIFO-Status“ des 8279 löschen
07D3  D3EF               OUT   EF
07D5  3E40               MVI   A,40   ;Kommandowort „FIFO-RAM lesen“
                                      zum 8279
07D7  D3EF               OUT   EF     ;Ausgeben
07D9  DBEF      WAREIN:  IN    EF     ;„FIFO-Status“ des 8279 lesen
07DB  E60F               ANI   F      ;Tastatur-Eingabe?
                                      ;/Bit3-0 = Anzahl der Zeichen im FIFO >< 0?
07DD  CAD907             JZ    WAREIN ;Nein!  /Warten
07E0  DBEE               IN    EE     ;Ja!  Eingegebenes Zeichen einlesen
07E2  C9                 RET           ;Rücksprung
```

A.3 Quellenhinweise

1. Bedienungsanleitung Experimentiercomputer ECB85
   Bestell-Nr. B2078 Siemens AG
2. Folienserie Speicher
3. Folienserie Aufbau und Wirkungsweise von Mikroprozessoren
   Wennar
4. Einführung in die Microprozessoren und Microcomputer
   Rudolf Schief ATTEMP TO Verlag GmbH Tübingen
5. Handbuch der digitalen Schaltungen
   Zuiderveen Emmo A. Franzis-Verlag
6. Assembler Programmiersprache System SAB 8080
   Siemens AG Bestell-Nr. B18814
7. Technische Mitteilungen an dem Bereich Bauelemente
   SAB8085 Microcomputersystem Bestell-Nr. B1854
   Siemens AG
8. ISIS II
   8080/8085 SME Tabellenheft Siemens AG
9. Mikrocomputer-Bausteine
   Mikroprozessorsystem SAB8085
   Datenbuch 1980/81 Siemens AG

A.4 Befehlsliste des 8085

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits	Bytes	Taktzyklen 8080/8085		Englische Befehlsbeschreibung	Funktion des Befehls
TRANSFERBEFEHLE							
a) Register → Register							
MOV r1, r2	01dddsss	- - - - -	1	5	4	Move register to register	r1, r2 = A, B, C, D, E, H, oder L: Lade Register r1 mit dem Inhalt von Register r2
XCHG	11101011	- - - - -	1	4	4	Exchange D & E, H & L	Vertausche Inhalte der Registerpaare (D, E) und (H, L)
XTHL	11100011	- - - - -	1	18	16	Exchange top of stack H & L	Vertausche Inhalt des Registerpaares (H, L) und den Inhalt des Wortes, das durch den Stackpointer adressiert ist
SPHL	11111001	- - - - -	1	5	6	H & L to stackpointer	Lade Stackpointer mit dem Inhalt des Registerpaares (H, L)
b) Speicher, Peripherie → Register							
MOV r1, M	01ddd110	- - - - -	1	7	7	Move memory to register	r1 = A, B, C, D, E, H oder L: Lade Register r1 mit dem Inhalt des Speicherbytes, das durch den Inhalt des Registerpaares (H, L) adressiert ist
LDA adr	00111010	- - - - -	3	13	13	Load accu direct	Akkumulator laden mit dem Inhalt der Adresse adr
LDAX rp	00rr1010	- - - - -	1	7	7	Load accu indirect	rp = B, D: Akkumulator laden mit dem Inhalt des Speicherplatzes, der durch den Inhalt des Registerpaares rp adressiert ist
LHLD adr	00101010	- - - - -	3	16	16	Load H & L direct	Lade Registerpaar (H, L) mit dem Inhalt der Adressen adr und (adr+1)
POP rp PSW	11rr0001 11110001	- - - - - Z, S, P, CY, AC	1	10	10	Pop register pair off stack	rp = B, D, H, PSW: Registerpaar rp wird mit dem Wort geladen, das durch den Stackpointer adressiert ist
IN nr	11011011	- - - - -	2	10	10	Input	Akkumulator wird mit dem Inhalt des Eingabekanals (Nummer nr ≦ 255) geladen
c) Konstante → Registerpaar							
LXI rp, adr	00rr0001	- - - - -	3	10	10	Load register pair immediate	rp = B, D, H, SP: Lade Registerpaar rp mit Wert adr
d) Register → Speicher, Peripherie							
MOV M, r1	01110sss	- - - - -	1	7	7	Move register to memory	r1 = A, B, C, D, E, H oder L: Inhalt von Register r1 auf den Speicherplatz abspeichern, der durch den Inhalt des Registerpaares (H, L) adressiert ist
STA adr	00110010	- - - - -	3	13	13	Store accu direct	Akkumulator-Inhalt unter Adresse adr abspeichern
STAX rp	00rr0010	- - - - -	1	7	7	Store accu indirect	rp = B, D: Akkumulator in dem Byte abspeichern, das durch den Inhalt des Registerpaares rp adressiert ist
SHLD adr	00100010	- - - - -	3	16	16	Store H & L direct	Registerpaar (H, L) unter Adresse adr und (adr+1) abspeichern
PUSH rp	11rr0101	- - - - -	1	11	12	Push register pair rp on stack	rp = B, D, H, PSW: Inhalt des Registerpaares rp wird in das Wort übertragen, das durch den Stackpointer adressiert ist
OUT nr	11010011	- - - - -	2	10	10	Output	Akkumulator-Inhalt wird auf Ausgabekanal (Nummer nr ≦ 255) ausgegeben
e) Konstante → Register-Speicher							
MVI M, konst	00110110	- - - - -	2	10	10	Move to memory immediate	Lade den Speicherplatz, der durch den Inhalt des Registerpaares (H, L) adressiert ist, mit Konstante (konst ≦ 255)
MVI r1, konst	00ddd110	- - - - -	2	7	7	Move immediate Register	r1 = A, B, C, D, E, H oder L: Lade Register r1 mit Konstante (konst ≦ 255)

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits	Bytes	Taktzyklen 8080/8085		Englische Befehlsbeschreibung	Funktion des Befehls

ARITHMETISCHE OPERATIONEN

INR r1	00ddd100	Z, S, P, - AC	1	5	4	Increment register	r1 = A, B, C, D, E, H oder L: Zum Inhalt des Registers r1 wird 1 addiert
INR M	00110100	Z, S, P, - AC	1	10	10	Increment memory	Zum Inhalt des durch Registerpaar (H, L) adressierten Bytes wird 1 addiert
DCR r1	00ddd101	Z, S, P, - AC	1	5	4	Decrement register	r1 = A, B, C, D, E, H oder L: Vom Inhalt des Registers r1 wird 1 subtrahiert
DCR M	00110101	Z, S, P, - AC	1	10	10	Decrement memory	Vom Inhalt des durch Registerpaar (H, L) adressierten Bytes wird 1 subtrahiert
INX rp	00rr0011	- - - - -	1	5	6	Increment register pair	rp = B, D, H, SP: Der Inhalt des Registerpaares rp wird um 1 erhöht
DCX rp	00rr1011	- - - - -	1	5	6	Decrement register pair	rp = B, D, H, SP: Der Inhalt des Registerpaares rp wird um 1 erniedrigt
ADD r1	10000sss	Z, S, P, CY, AC	1	4	4	Add register to accu	r1 = A, B, C, D, E, H oder L: Inhalt von Register r1 wird zum Inhalt des Akkumulators addiert
ADD M	10000110	Z, S, P, CY, AC	1	7	7	Add memory to accu	Inhalt des Speicherbytes, das durch den Inhalt des Registerpaares (H, L) adressiert ist, wird zum Inhalt des Akkumulators addiert
ADC r1	10001sss	Z, S, P, CY, AC	1	4	4	Add register to accu with carry	r1 = A, B, C, D, E, H oder L: Inhalt von Register r1 und Inhalt des Carry-Bits werden zum Inhalt des Akkumulators addiert
ADC M	10001110	Z, S, P, CY, AC	1	7	7	Add memory to accu with carry	Inhalt des Speicherbytes, das durch den Inhalt des Registerpaares (H, L) adressiert ist und der Inhalt des Carry-Bits werden zum Inhalt des Akkumulators addiert
DAD rp	00rr1001	- - - CY -	1	10	10	Add register pair to H and L	rp = B, D, H, SP: Inhalt des Registerpaares rp und der Inhalt des Registerpaares (H, L) werden addiert. Ergebnis in (H, L)
SUB r1	10010sss	Z, S, P, CY, AC	1	4	4	Subtract register from accu	r1 = A, B, C, D, E, H oder L: Inhalt des Registers r1 wird vom Akkumulator-Inhalt subtrahiert
SUB M	10010110	Z, S, P, CY, AC	1	7	7	Subtract memory from accu	Inhalt des Speicherbytes, das durch den Inhalt des Registerpaares (H, L) adressiert ist, wird vom Akkumulator subtrahiert
SBB r1	10011sss	Z, S, P, CY, AC	1	4	4	Subtract register from accu with borrow	r1 = A, B, C, D, E, H oder L: Inhalt von Register r1 und Inhalt des Carry-Bits werden vom Akkumulator-Inhalt subtrahiert
SBB M	10011110	Z, S, P, CY, AC	1	7	7	Subtract memory from accu with borrow	Inhalt des Speicherbytes, das durch das Registerpaar (H, L) adressiert ist und Inhalt des Carry-Bits werden vom Akkumulator subtrahiert
ADI konst	11000110	Z, S, P, CY, AC	2	7	7	Add immediate to accu	Konstante (konst ≦ 255) wird zum Inhalt des Akkumulators addiert
ACI konst	11001110	Z, S, P, CY, AC	2	7	7	Add immediate to accu with carry	Zum Akkumulator-Inhalt werden konst ≦ 255 und Carry-Bit addiert
SUI konst	11010110	Z, S, P, CY, AC	2	7	7	Subtract immediate from accu	konst ≦ 255 wird vom Akkumulator-Inhalt subtrahiert
SBI konst	11011110	Z, S, P, CY, AC	2	7	7	Subtract immediate from accu with borrow	konst ≦ 255 und das Carry-Bit werden vom Akkumulator-Inhalt subtrahiert
DAA	00100111	Z, S, P, CY, AC	1	4	4	Decimal adjust accu	Akkumulator-Inhalt wird in eine 2stellige Zahl umgewandelt

LOGISCHE OPERATIONEN

CMA	00101111	- - - - -	1	4	4	Complement accu	Akkumulator-Inhalt wird negiert
ANA r1 für 8085	10100sss	Z, S, P, 0, AC Z, S, P, 0, 1	1	4	4	And register with accu	r1 = A, B, C, D, E, H oder L: Akkumulator-Inhalt und der Inhalt des Registers r1 werden UND-verknüpft
ANA M für 8085	10100110	Z, S, P, 0, AC Z, S, P, 0, 1	1	7	7	And memory with accu	Der Inhalt des durch Registerpaar (H, L) adressierten Bytes wird mit dem Akkumulator-Inhalt UND-verknüpft
ANI konst für 8085	11100110	Z, S, P, 0, AC Z, S, P, 0, 1	2	7	7	And immediate with accu	Akkumulator-Inhalt wird mit der konst ≦ 255 UND-verknüpft
ORA r1	10110sss	Z, S, P, 0, 0	1	4	4	Or register with accu	r1 = A, B, C, D, E, H oder L: Akkumulator-Inhalt wird mit dem Inhalt des Registers r1 ODER-verknüpft
ORA M	10110110	Z, S, P, 0, 0	1	7	7	Or memory with accu	Inhalt des über Registerpaar (H, L) adressierten Bytes wird mit dem Akkumulator-Inhalt ODER-verknüpft
ORI konst	11110110	Z, S, P, 0, 0	2	7	7	Or immediate with accu	Akkumulator-Inhalt wird mit konst ≦ 255 ODER-verknüpft

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits	Bytes	Taktzyklen 8080/8085		Englische Befehlsbeschreibung	Funktion des Befehls
XRA ri	10101sss	Z, S, P, 0, 0	1	4	4	Exclusive Or register with accu	ri = A, B, C, D, E, H oder L: Akkumulator-Inhalt wird mit dem Inhalt des Registers ri Exclusive-ODER-verknüpft
XRA M	10101110	Z, S, P, 0, 0	1	7	7	Exclusive Or memory with accu	Das über Registerpaar (H, L) adressierte Byte wird durch Exclusive-ODER mit dem Akkumulator verknüpft
XRI konst	11101110	Z, S, P, 0, 0	2	7	7	Exclusive Or immediate with accu	Der Akkumulator wird mit dem Wert konst ≦ 255 durch Exclusive-ODER verknüpft
CMP ri	10111sss	Z, S, P, CY, AC	1	4	4	Compare register with accu	ri = A, B, C, D, E, H oder L: Akkumulator-Inhalt wird mit dem Inhalt des Registers ri verglichen
CMP M	10111110	Z, S, P, CY, AC	1	7	7	Compare memory with accu	Akkumulator-Inhalt wird mit dem Inhalt des durch Registerpaar (H, L) adressierten Bytes verglichen
CPI konst	11111110	Z, S, P, CY, AC	2	7	7	Compare immediate with accu	Akkumulator-Inhalt wird mit konst ≦ 255 verglichen

REGISTERANWEISUNGEN

a) Akkumulator rotieren

RLC	00000111	- - - CY -	1	4	4	Rotate accu left	Akkumulator-Inhalt wird zyklisch um 1 Bit nach links verschoben. Bit 2^7 wird in das Carry-Bit geschrieben. Bit 2^0 = Bit 2^7
RRC	00001111	- - - CY -	1	4	4	Rotate accu right	Akkumulator-Inhalt wird zyklisch um 1 Bit nach rechts verschoben. Bit 2^0 wird in das Carry-Bit geschrieben. Bit 2^7 = Bit 2^0
RAL	00010111	- - - CY -	1	4	4	Rotate accu left through carry	Akkumulator-Inhalt wird um 1 Bit nach links geschoben. Bit 2^7 wird in das Carry-Bit und das Carry-Bit in das Bit 2^0 geschrieben
RAR	0001111	- - - CY -	1	4	4	Rotate accu right through carry	Akkumulator-Inhalt wird um 1 Bit nach rechts geschoben. Bit 2^0 wird in das Carry-Bit und das Carry-Bit in das Bit 2^7 geschrieben

b) Übertragungsbit-Anweisungen

CMC	00111111	- - - CY -	1	4	4	Complement carry	Carry-Bit wird negiert
STC	00110111	- - - 1 -	1	4	4	Set carry	Carry-Bit wird gesetzt

SPRUNGBEFEHLE

a) Unbedingte Sprünge

PCHL	11101001	- - - - -	1	5	6	H & L to program counter	Programm wird an der Adresse fortgesetzt, die im Registerpaar (H, L) steht
JMP adr	11000011	- - - - -	3	10	10	Jump unconditional	Programm wird an der Adresse adr fortgesetzt

b) Bedingte Sprünge

JC adr	11011010	- - - - -	3	10	7/10	Jump on carry	Bei Carry-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt
JNC adr	11010010	- - - - -	3	10	7/10	Jump on no carry	Bei Carry-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt
JZ adr	11001010	- - - - -	3	10	7/10	Jump on zero	Bei Zero-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt
JNZ adr	11000010	- - - - -	3	10	7/10	Jump on no zero	Bei Zero-Bit = 0 wird das Programm bei der Adresse fortgesetzt
JM adr	11111010	- - - - -	3	10	7/10	Jump on minus	Bei Sign-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt
JP adr	11110010	- - - - -	3	10	7/10	Jump on positiv	Bei Sign-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt
JPE adr	11101010	- - - - -	3	10	7/10	Jump on parity even	Bei Parity-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt
JPO adr	11100010	- - - - -	3	10	7/10	Jump on parity odd	Bei Parity-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits	Bytes	Taktzyklen 8080/8085	Englische Befehlsbeschreibung	Funktion des Befehls

UNTERPROGRAMMBEHANDLUNG

a) Programmaufrufe

Bei allen Aufrufbefehlen wird die Rückkehradresse in dem Wort, das durch den Stackpointer adressiert ist, abgelegt

CALL adr	11001101	- - - - -	3	17 18	Call unconditional	Programm wird bei der Adresse fortgesetzt
CC adr	11011100	- - - - -	3	11/17 9/18	Call on carry	Bei Carry-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt
CNC adr	11010100	- - - - -	3	11/17 9/18	Call on no carry	Bei Carry-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt
CZ adr	11001100	- - - - -	3	11/17 9/18	Call on zero	Bei Zero-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt
CNZ adr	11000100	- - - - -	3	11/17 9/18	Call on no zero	Bei Zero-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt
CM adr	11111100	- - - - -	3	11/17 9/18	Call on minus	Bei Sign-Bit = 1 wird das Programm bei der Adresse adr fortgesetzt
CP adr	11110100	- - - - -	3	11/17 9/18	Call on positiv	Bei Sign-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt
CPE adr	11101100	- - - - -	3	11/17 9/18	Call on parity even	Bei Parity-Bit = 1 wird das Program bei der Adresse adr fortgesetzt
CPO adr	11100100	- - - - -	3	11/17 9/18	Call on parity odd	Bei Parity-Bit = 0 wird das Programm bei der Adresse adr fortgesetzt
RST konst	11nnn111	- - - - -	1	11 12	Restart	Programm wird auf der Adresse 8 x konst fortgesetzt ($0 \leqq$ konst $\leqq 7$)

b) Rücksprungbefehle

RET	11001001	- - - - -	1	10 10	Return	Programm wird an der Adresse fortgesetzt, die in dem Wort steht, das über dem Stackpointer adressiert ist
RC	11011000	- - - - -	1	5/11 6/12	Return on carry	Bei Carry-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht
RNC	11010000	- - - - -	1	5/11 6/12	Return on no carry	Bei Carry-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht
RZ	11001000	- - - - -	1	5/11 6/12	Return on zero	Bei Zero-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht
RNZ	11000000	- - - - -	1	5/11 6/12	Return on no zero	Bei Zero-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht
RM	11111000	- - - - -	1	5/11 6/12	Return on minus	Bei Sign-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht
RP	11110000	- - - - -	1	5/11 6/12	Return on positiv	Bei Sign-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht
RPE	11101000	- - - - -	1	5/11 6/12	Return pn parity even	Bei Parity-Bit = 1 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht
RPO	11100000	- - - - -	1	5/11 6/12	Return on parity odd	Bei Parity-Bit = 0 wird das Programm an der Adresse fortgesetzt, die in dem über dem Stackpointer adressierten Wort steht

PROGRAMMUNTERBRECHUNG

EI	11111011	- - - - -	1	4 4	Enable interrupts	INTE-Flipflop wird gesetzt; der Mikroprozessor kann eine Unterbrechungsanforderung annehmen
DI	11110011	- - - - -	1	4 4	Disable interrupts	INTE-Flipflop wird rückgesetzt; der Mikroprozessor ignoriert Unterbrechungsanforderungen

SONSTIGE BEFEHLE

HLT	01110110	- - - - -	1	7 5	Halt	Programm hält an, bis eine Unterbrechungsanforderung eintritt
NOP	00000000	- - - - -	1	4 4	No operation	Leerbefehl

Mnemonic	Binär-Code	Beeinflußte Zustands-Bits	Bytes	Taktzyklen 8080/8085	Englische Befehlsbeschreibung	Funktion des Befehls
SAB 8085 BEFEHLE						
RIM	00100000	- - - - -	1	– 4	Read interrupt Mask	Lesen Unterbrechungsmaske und seriellen Eingang in Akkumulator
SIM	00110000	- - - - -	1	– 4	Set interrupt Mask	Setzen Unterbrechungsmaske und seriellen Ausgang

Befehlsliste in Hex-Folge

Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic
00	NOP	2B	DCX H	56	MOV D, M	81	ADD C	AC	XRA H	D7	RST 2
01	LXI B, D16	2C	INR L	57	MOV D, A	82	ADD D	AD	XRA L	D8	RC
02	STAX B	2D	DCR L	58	MOV E, B	83	ADD E	AE	XRA M	D9	- - -
03	INX B	2E	MVI L, D8	59	MOV E, C	84	ADD H	AF	XRA A	DA	JC Adr
04	INR B	2F	CMA	5A	MOV E, D	85	ADD L	B0	ORA B	DB	IN D8
05	DCR B	30	SIM	5B	MOV E, E	86	ADD M	B1	ORA C	DC	CC Adr
06	MVI B, D8	31	LXI SP, D16	5C	MOV E, H	87	ADD A	B2	ORA D	DD	- - -
07	RLC	32	STA Adr	5D	MOV E, L	88	ADC B	B3	ORA E	DE	SBI D8
08	- - -	33	INX SP	5E	MOV E, M	89	ADC C	B4	ORA H	DF	RST 3
09	DAD B	34	INR M	5F	MOV E, A	8A	ADC D	B5	ORA L	E0	RPO
0A	LDAX B	35	DCR M	60	MOV H, B	8B	ADC E	B6	ORA M	E1	POP H
0B	DCX B	36	MVI M, D8	61	MOV H, C	8C	ADC H	B7	ORA A	E2	JPO Adr
0C	INR C	37	STC	62	MOV H, D	8D	ADC L	B8	CMP B	E3	XTHL
0D	DCR C	38	- - -	63	MOV H, E	8E	ADC M	B9	CMP C	E4	CPO Adr
0E	MVI C, D8	39	DAD SP	64	MOV H, H	8F	ADC A	BA	CMP D	E5	PUSH H
0F	RRC	3A	LDA Adr	65	MOV H, L	90	SUB B	BB	CMP E	E6	ANI D8
10	- - -	3B	DCX SP	66	MOV H, M	91	SUB C	BC	CMP H	E7	RST 4
11	LXI D, D16	3C	INR A	67	MOV H, A	92	SUB D	BD	CMP L	E8	RPE
12	STAX D	3D	DCR A	68	MOV L, B	93	SUB E	BE	CMP M	E9	PCHL
13	INX D	3E	MVI A, D8	69	MOV L, C	94	SUB H	BF	CMP A	EA	JPE Adr
14	INR D	3F	CMC	6A	MOV L, D	95	SUB L	C0	RNZ	EB	XCHG
15	DCR D	40	MOV B, B	6B	MOV L, E	96	SUB M	C1	POP B	EC	CPE Adr
16	MVI D, D8	41	MOV B, C	6C	MOV L, H	97	SUB A	C2	JNZ Adr	ED	- - -
17	RAL	42	MOV B, D	6D	MOV L, L	98	SBB B	C3	JMP Adr	EE	XRI D8
18	- - -	43	MOV B, E	6E	MOV L, M	99	SBB C	C4	CNZ Adr	EF	RST 5
19	DAD D	44	MOV B, H	6F	MOV L, A	9A	SBB D	C5	PUSH B	F0	RP
1A	LDAX D	45	MOV B, L	70	MOV M, B	9B	SBB E	C6	ADI D8	F1	POP PSW
1B	DCX D	46	MOV B, M	71	MOV M, C	9C	SBB H	C7	RST 0	F2	JP Adr
1C	INR E	47	MOV B, A	72	MOV M, D	9D	SBB L	C8	RZ	F3	DI
1D	DCR E	48	MOV C, B	73	MOV M, E	9E	SBB M	C9	RET	F4	CP Adr
1E	MVI E, D8	49	MOV C, C	74	MOV M, H	9F	SBB A	CA	JZ Adr	F5	PUSH PSW
1F	RAR	4A	MOV C, D	75	MOV M, L	A0	ANA B	CB	- - -	F6	ORI D8
20	RIM	4B	MOV C, E	76	HLT	A1	ANA C	CC	CZ Adr	F7	RST 6
21	LXI H, D16	4C	MOV C, H	77	MOV M, A	A2	ANA D	CD	CALL Adr	F8	RM
22	SHLD Adr	4D	MOV C, L	78	MOV A, B	A3	ANA E	CE	ACI D8	F9	SPHL
23	INX H	4E	MOV C, M	79	MOV A, C	A4	ANA H	CF	RST 1	FA	JM Adr
24	INR H	4F	MOV C, A	7A	MOV A, D	A5	ANA L	D0	RNC	FB	EI
25	DCR H	50	MOV D, B	7B	MOV A, E	A6	ANA M	D1	POP D	FC	CM Adr
26	MVI H, D8	51	MOV D, C	7C	MOV A, H	A7	ANA A	D2	JNC Adr	FD	- - -
27	DAA	52	MOV D, D	7D	MOV A, L	A8	XRA B	D3	OUT D8	FE	CPI D8
28	- - -	53	MOV D, E	7E	MOV A, M	A9	XRA C	D4	CNC Adr	FF	RST 7
29	DAD H	54	MOV D, H	7F	MOV A, A	AA	XRA D	D5	PUSH D		
2A	LHLD Adr	55	MOV D, L	80	ADD B	AB	XRA E	D6	SUI D8		

D8 = Eine Konstante oder ein logisch/arithmetischer Ausdruck, der eine 8 Bit Datengröße darstellt.

D16 = Eine Konstante oder ein logisch/arithmetischer Ausdruck, der eine 16 Bit Datengröße darstellt.

Adr = Eine 16 Bit Adresse

Eine Einführung in das Experimentieren mit Mikrocomputern der 8085-Familie, auf der Basis der Sendereihe „Mikroprozessoren" im 3. Fernsehprogramm des NDR.

Mit diesem Buch bekommt der Leser eine Hilfe in die Hand, mit der er aus seinem Mikrocomputer mehr machen kann.

Zu Beginn werden die *Komponenten* eines Mikrocomputersystems ausführlich erläutert, und es wird das Prinzip der *Programmierung* behandelt. Dabei wird auf den Experimentiercomputer ECB 85 Bezug genommen; die Maschinenbefehle lassen sich aber auch auf andere 8080- oder 8085-Mikrocomputer übertragen.

Im zweiten Teil des Buches werden zahlreiche *Versuche* beschrieben; dies geschieht meist in Form von Übungen, so daß der Leser die Ergebnisse nach und nach selbst erarbeiten kann, bis das fertige Programm läuft und die gewünschten Steuerungen vornimmt.

Als Beispiele für die ersten vorgestellten Anwendungen seien hier genannt: das Prinzip der Multiplikation mit einem Mikrocomputer; das Heraussuchen des größten (oder kleinsten) Wertes aus Datenlisten; die Ansteuerung von Ziffernanzeigen; der Mikrocomputer als Uhr.

Die *Hardware-Anwendungen* schließlich sind sehr praxisnah konzipiert und auch in manchen Modellanlagen verwendbar: Ampelsteuerung; Dreiklang-Erzeugung, bis hin zu kompletten Melodien; und schließlich die Geschwindigkeitsmessung z. B. an einer Modellrennbahn. Die gezeigten Beispiele lassen sich vielfältig variieren und erweitern.
Im Anhang sind ausführliche Befehlslisten enthalten.

ISBN 3-8025-1229-4